

Настройка и программирование на STL контроллеров SIMATIC в Step7

Учебное пособие по курсу
**"АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ
ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ"**

для специальности 230102 «Автоматизированные системы обработки информации и управления» (специализация АСРВ)

Составил доц. Дубина С.М.

Самара – 2008

1. Введение

Слово «архитектура» в названии курса означает «структура», а контроллер – специализированный микропроцессор, применяемый в автоматических системах (АС). Таким образом в данном курсе рассматривается структура и программирование для АС контроллеров фирмы Siemens.

АС – обобщённое название систем автоматического управления (например, летательными аппаратами) и систем автоматического регулирования (САР). Каждая из этих систем имеет свои особенности, но их объединяет общая идея и схема функционирования.

1.1. Общая схема САР

Укрупнённая схема САР показана на рис.1.1. Её основными частями являются : объект управления (ОУ), установленные на ОУ датчики, система регулирования и установленные на ОУ исполнительные устройства (ИУ).

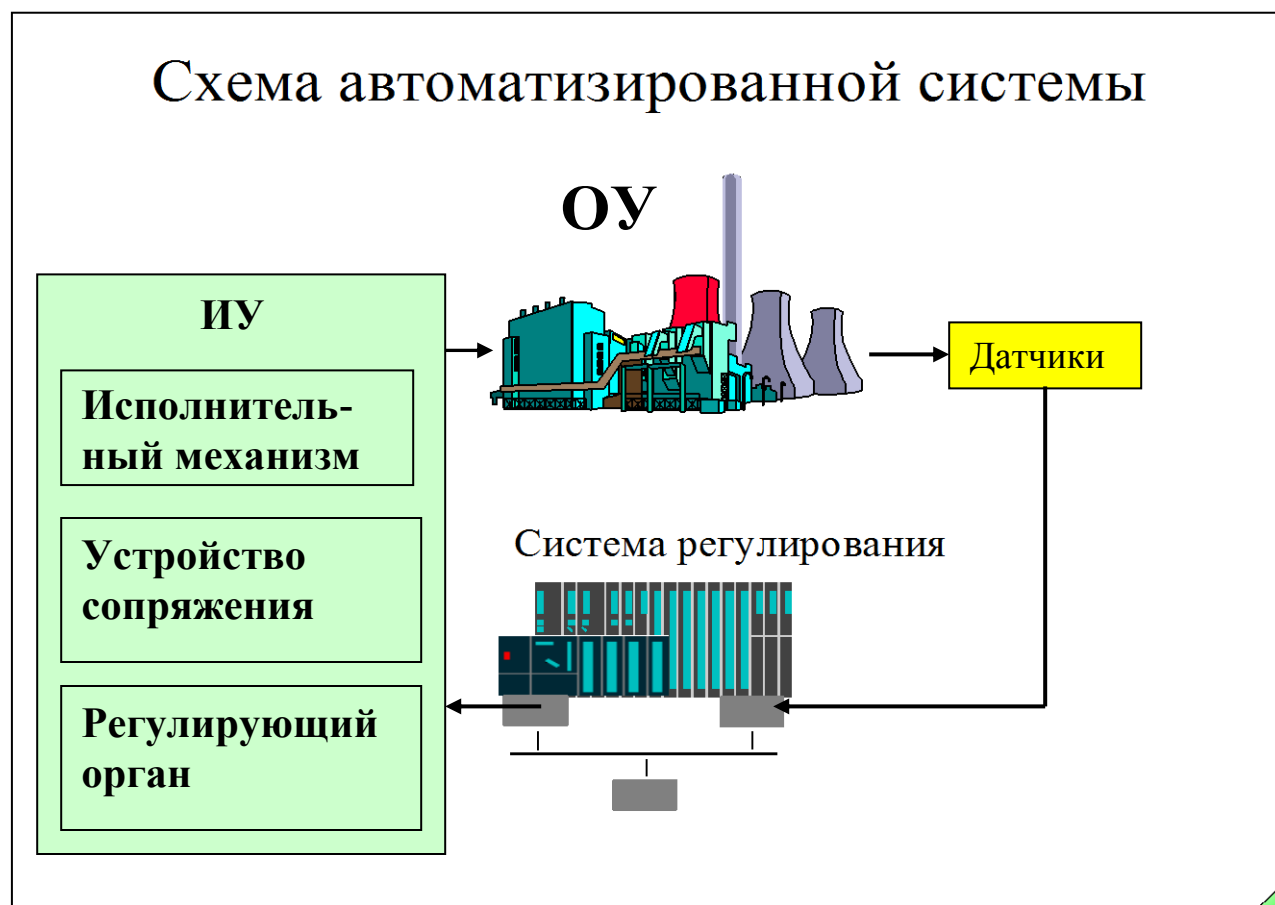


Рис. 1.1.

Объектом управления может быть технологический процесс или техническая система. Датчики, система регулирования и ИУ выбираются исходя из задач управления/регулирования и свойств ОУ.

1.2. Процессы и их характеристики

Управление процессами ¹⁾ и их параметрами в промышленности является главной областью применения регулирующих приборов и устройств ²⁾. В основном они существуют в форме

отдельных регулирующих приборов или программных регуляторов и применяются в машиностроении для регулирования температуры, давления, потока жидкости, количества, дозирования, уровня, положения (позиции), скорости, расстояния, а также в технологии производства для регулирования концентрации, проводимости, вязкости, плотности, химического состава и т. д.

Такое широкое поле применений увеличивается еще больше за счет того, что почти в каждом регулировании существуют еще и различные производственные требования, которые делают классификацию области применения почти невозможной. В основном различают непрерывные производственные процессы и не непрерывные производственные процессы (загрузочные процессы, рецептурные технологии и т. д.), которые называют также прерывистыми, или событийно-дискретными процессами (поштучными процессами).

Почти каждое регулирование определенной физической величины дает свои собственные специфические проблемы, которые зачастую определяются применяемыми исполнительными устройствами и датчиками (сенсорами) ³⁾.

1.2.1. Температура

Управление производственными участками с температурным регулированием причисляется и сегодня (часто несправедливо) к “простейшим” задачам регулирования. Оно в большинстве случаев является несимметричным (с различным поведением при нагреве и при охлаждении) и имеет нелинейную характеристику для применяемых исполнительных и сенсорных устройств. Следствием является медленный ввод в действие из-за длительного нагрева и охлаждения, а также из-за влияния на процесс термического выравнивания соседних производственных участков с температурным регулированием. Эти процессы затрудняют оптимизацию и отладку регуляторов. Хотя температурное регулирование относят к медленным процессам, оптимизация комплексных температурных участков тяжела, а внешние средства оптимизации на практике редко дают приемлемые результаты. Большинство термических процессов в станкостроении и в машиностроении оборудуются компактными регуляторами или программно реализованными регулируемыми установками, не в последнюю очередь из соображений стоимости.

Для измерения регулируемых температур применяются резисторные термометры, термоэлементы и термометры на основе излучения (пирометры).

1.2.2. Давление

Регулировки давления считаются вообще быстрыми процессами, которые - в зависимости от обстоятельств - могут находиться на границе (по быстрдействию) компактных регулирующих приборов и программно реализованных установок. По методам различают измерения давления и разности давлений

В технологии применяют главным образом статические системы измерения давления в жидких или газообразных средах. Участки с регулировкой давления в большинстве случаев являются чисто инерционными звеньями невысокого порядка и поэтому легко оптимизируются. Элементы оптимизации дают, как правило, очень хорошие результаты.

Критичным для регулятора давления может быть случай, когда запирающий механизм

1) Под процессом здесь понимается ограниченная система, в которой преобразуется и/или передается вещество, энергия и/или информация.

2) Регулирующие приборы в общем случае включают и программное обеспечение. Оно охватывает по меньшей мере измерительный орган для управления регулируемой величиной, сравнивающее устройство для получения отклонения в регулировании, вычислитель для определения отклонения от установленного значения, которое отрабатывается в регулирующем процессе.

3) Технические устройства, которые воспринимают измеряемые значения. Сенсорами называют миниатюрные приемники значений измеряемых величин, снабженные интегральными электронными схемами (усилителями).

в подводящей системе должен реагировать быстро. Чтобы отрабатывать ударные повышения давления, предъявляются очень высокие требования к динамике регулятора и его исполнительных органов.

Измерение давления выполняется индуктивными, пьезоэлектрическими или емкостными методами с помощью омических тензометрических линий (DMS – Dehnungsmessstreifen) или резонансными проволочными датчиками.

1.2.3. Поток и количество

Регулирование потока и количества играет важную роль в технологических установках; сточки зрения техники регулирования оно ведёт себя на практике подобно регулированию давления, хотя управление паузами может вызвать затруднения. Предпосылкой хорошего регулирования потока или количества является качественное и чувствительное исполнение датчиков (сенсоров). При регулировании потока жидкости на практике обычно возникают проблемы с датчиками, которые часто неправильно нагружают регулятор. Для измерения количества, как правило, учитывают величину потока и время.

Для измерения потока или количества проводят измерения действующего давления, подсчет перемещений, подсчет потока, измерение потока плавающих частиц, магнито-индуктивные измерения потока (MID – magnetisch-induktive Durchmessungen), применяются кориолисовы измерители потока (CMD-Coriolis-Durchflussmessung) и ультразвуковые измерители (USD – Ultraschall-Durchflussmessungen). Современные измерители потока или количества часто выполняются как самостоятельные переносные приборы с собственным вычислителем и предоставляют комфортабельные вспомогательные устройства как для соответствующего нормирования измерительных сигналов, так и для предварительной индикации измеренных значений.

1.2.4. Дозирование

Дозирование – это отмеривание, выдача или отделение определенного количества. В технологии дозирование часто связано с рецептами. Оно может выполняться прерывисто или непрерывно. По определению регулирования, в смысле замкнутого рабочего контура, задачей дозирования является регулирование потока материала, и этим оно очень сходно с регулированием давления или количества. При регулировании дозирования происходит установленное воздействие путем добавки жидкой или твердой компоненты к процессу загрузки или же к непрерывному процессу. Эта задача выполняется дозирующими насосами, дозирующими ленточными конвейерами, барабанными, вибраторными и шнековыми дозаторами.

Для измерения регулируемых количеств используются упомянутые выше датчики давления и количества.

1.2.5. Уровень

Регулирование уровня в технологии часто имеет место в его чистом виде как просто управляемое и легко выполняемое с помощью P(D) регуляторов (т. е. пропорциональных или дифференциальных регуляторов). На практике вследствие ряда учитываемых возмущений могут быть поставлены отдельные высокие требования при проектировании концепции регулирования.

1.2.6. Положение, позиция, скорость и расстояние

Положение, позиция, скорость и расстояние являются уже предельными случаями многих стандартных регулирующих приборов. Чистое регулирование положения в динамичной системе часто не может быть выполнено из-за скоростных ограничений. Исключением являются задачи

медленного позиционирования, такие как компенсационное, позиционное регулирование валков и т.д. Различают измерения положения (позиции) и расстояния, всех их охватывают угловыми шаговыми датчиками, **SSI-датчиками**, импульсными датчиками и реверсивными счетчиками.

1.2.7. Химические параметры

Под регулированием химических параметров часто подразумевается регулирование концентрации, как, например, значения pH, кислотности и других биохимических измеряемых величин, которые могут быть получены только очень затратными методами анализа. Нередко при таких биохимических методах важную роль играют термические связи, как, например, температурное регулирование экзотермической реакции или температурное регулирование в колонке. Эти типы регулирования относятся в общем случае к задачам комплексного регулирования в области технологии, так как эти процессы часто очень нелинейны, изменчивы во времени, имеют паузы и/или несимметричны.

1.3. Регуляторы и их классификация

Регуляторы – в смысле собственно регулирования – есть функциональные элементы, которые в зависимости от измеренной технически (сенсором) величины процесса воздействуют по математически точному правилу на физическую величину в замкнутом контуре с помощью активного органа. Это показано на рис. 1.2.

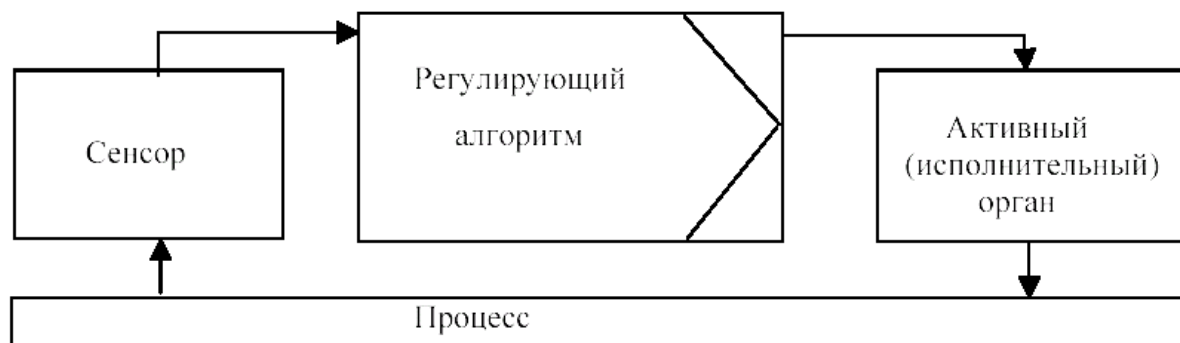


Рис. 1.2.

Практически регуляторы содержат не только вычислительное правило (алгоритм), но имеют и ряд управляющих функций для обслуживания, наблюдения, обеспечения безопасности и возможностей переключений в контуре управления.

Все регуляторы (PID – регуляторы, т. е. состоящие из звеньев пропорциональной передачи, интегрирующих и дифференцирующих), по существу делятся по назначению: образование входного сигнала, формирование требуемого значения, формирование регулирующей разности, алгоритм (P, I, D, Z – звенья) и формирование выходного сигнала. Для каждого звена регулятора есть ряд различных дополнительных – по выбору – функций, применение которых существенно влияет на характеристики регулятора и определяет классификацию множества регуляторов и их типов.

Наряду с указанными возможностями классифицировать регуляторы и регулирующие устройства по их параметрам имеются и другие возможности их деления. Это многообразие различных понятий – зачастую с одинаковым значением – увеличивает предполагаемую сложность проблемной области “техника регулирования”.

В нормале DIN 19225 от 1981 года “Терминология и классификация регуляторов” сопоставляются подобные понятия. Слово “регулятор” можно сопоставить различные

дополнительные определения (например, температурный регулятор, компенсирующий регулятор, регулятор процесса, PI-регулятор, двухпозиционный регулятор, следящий регулятор, регулятор состояния и т. д.), и уже получается необозримое множество понятий, что часто и эксперты приходят в замешательство. Ниже приведена выборка примеров регуляторов и их деления.

Таблица 1.1. Обзор видов производимых регуляторов и их типичные представители в системе SIMATIC

	Регулятор				
	Промыш- ленный регулятор	Регулятор процесса	Универ- сальный регулятор	Специаль- ный, или отраслевой регулятор	Программный регулятор
Отдельный прибор	SIPART DR19	SIPART DR22, DR24	-		
Аппаратурная группа в SIMATIC S7			FM ¹⁾ 355 FM 455		
Аппаратурная группа в SIMATIC S5			IP ²⁾ 262	IP 244	
Программная группа в SIMATIC S7					PID Control Standard PID Control Modular PID Control
Программная группа в SIMATIC S5					OB251, R64 Regelung 100U Modulare Regelung

1) FM –сокращенное обозначение функционального модуля (конструктивно самостоятельный или встраиваемый модуль SIMATIC S7)

2) IP – сокращенное обозначение для интеллектуальной периферии (конструктивно самостоятельный или встраиваемый модуль SIMATIC S5)

1.4. Системы управления на базе ПЛК

В прошлом задачи управления решались с помощью автономных PLC (Programmable Logical Controls – программируемых логических контроллеров), управляющих машинами или процессами. Сегодня для того, чтобы оставаться конкурентоспособными, компаниям не достаточно автоматизировать только отдельные установки или машины. Требования большей гибкости в сочетании с высокой производительностью могут быть удовлетворены, когда отдельные машины и/или процессы будут объединены в единую автоматизированную систему.

Для комплексной автоматизации (TIA - Totally Integrated Automation) общей базой является общая среда программного обеспечения, которая объединяет все компоненты в единую систему, несмотря на разнообразие применяемых технологий. Она обеспечивает всем необходимым для программирования, конфигурирования, управления, обработки данных, коммуникаций и обслуживания всего проекта.

Программа-менеджер Step 7 SIMATIC Manager, запускаемая на ПК или на Siemens PG (программаторе), содержит встроенный набор утилит для всех системных компонентов, которые облегчают создание, тестирование, запуск, эксплуатацию и обслуживание проектов пользователя. Пока Вы программируете и конфигурируете свое решение системы управления, программное обеспечение Siemens сохраняет все Ваши данные в центральной базе данных, к которой все утилиты имеют доступ.

Центральная БД (Central Database) всех компонентов системы TIA – это база, данные в которую вводятся один раз и после этого являются доступными для всего проекта в целом.

Полная интеграция всех компонентов в единую систему автоматизации возможна с помощью:

- единой общей среды программного обеспечения (Step 7 SIMATIC Manager), которая объединяет все компоненты и задачи в унифицированную, удобную для использования систему;
- общей системы управления данными;
- стандартных открытых шин, таких как Ethernet, PROFIBUS, MPI, AS-interface, связывающих друг с другом все компоненты от уровня управления до полевого уровня.

1.5. SIMATIC S7

Общий обзор SIMATIC®

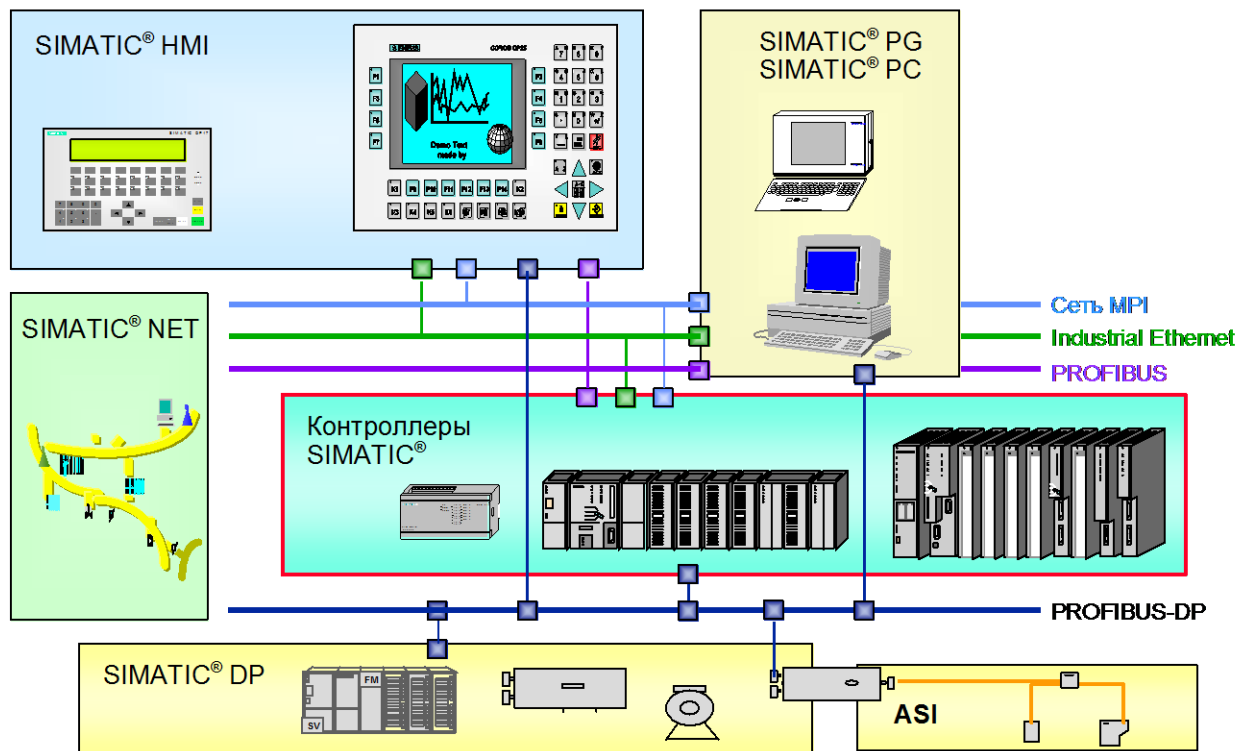


Рис. 1.3. Состав SIMATIC.

SIMATIC S7 имеет ряд серий, начинающихся с LOGO! – интеллектуального реле для коммутирования устройств (рис. 1.4)

Универсальные логические модули LOGO! являются компактными функционально законченными изделиями, предназначенными для построения наиболее простых устройств автоматического управления и могут иметь 34 встроенные функции, до 130 функций на программу. Логические модули, оснащённые дисплеем, могут программироваться со встроенной клавиатуры без использования программатора и специального программного обеспечения.

Семейство LOGO! объединяет в своем составе следующие объекты.

Универсальные логические модули :

- LOGO! Basic с встроенной клавиатурой и дисплеем ;
- LOGO! Pure без клавиатуры и дисплея .

Модули расширения :

- 8- и 16-канальные модули ввода - вывода дискретных сигналов DM8 и DM16;
- 2-канальные модули ввода аналоговых сигналов AM2 и AM2 PT100;
- коммуникационные модули для подключения к AS-Interface, LON и EIB.

Модули блоков питания LOGO! Power.

Модули LOGO! Contact для бесшумной коммутации 3-фазных цепей переменного тока .

Для более удобной разработки программ может использоваться пакет LOGO!SoftComfort.



Рис. 1.4. Модули LOGO!

Следующая серия - SIMATIC S7-200 (рис. 1.5). Это модульная малая система управления для задач наиболее низкого уровня производительности. Система рассчитана на использование CPU различной производительности (память до 8 кб, 8-40 интегрированных входов /выходов на CPU).

Каждый CPU имеет вариант питания либо от ≈ 24 В, либо $\sim 120/230$ В. В конструкцию может быть добавлено до 7 модулей расширения, в зависимости от CPU (кроме CPU 210 или CPU 221).

Примечание: Комбинация модулей серии S7-21x с модулями серии S7-22x не допускается.

CPU подключается к модулям гибкими встроенными прорезиненными кабелями (для S7-22x серии) и посредством шинных соединителей (для S7-21x серии).

Работа в сети использует коммуникационный интерфейс RS 485 (кроме CPU 210), в сети PROFIBUS S7-200 может использоваться как ведомое устройство (CPU 215, CPU 222 или мощнее).

Контроллер имеет дополнительно следующие свойства:

- возможно центральное подключение PG/ПК с доступом ко всем модулям,
- нет ограничений на использование слотов.

Контроллер использует собственное ПО - S7 Micro/WIN32, т.е. STEP 7 не требуется.

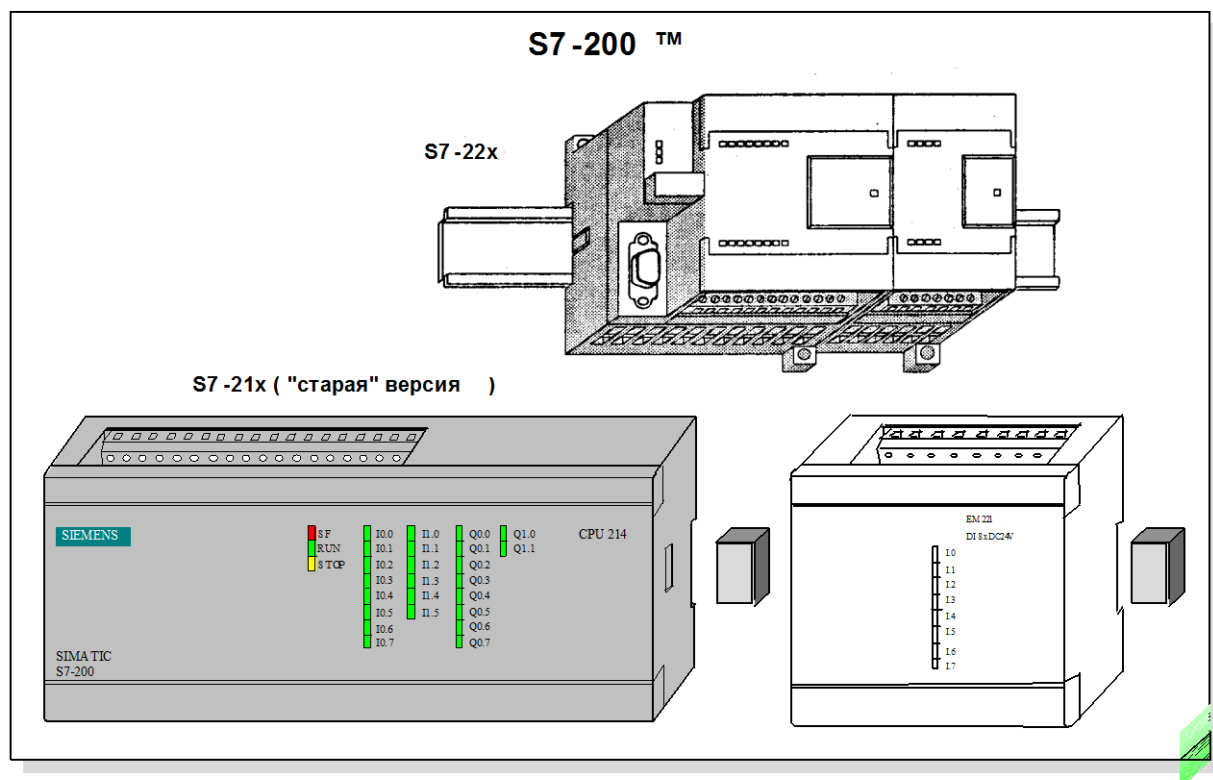


Рис. 1.5. S7-200

1.5.1. Структура программируемого контроллера S7-300/400

Следующий логический программируемый контроллер SIMATIC S7-300/400 имеет модульную конструкцию. Модули, из которых составляется требуемая конфигурация контроллера, могут быть центральными (располагаться по соседству с CPU) или распределенными. В системах SIMATIC S7 распределенные входы/выходы (I/O) являются составной частью системы. CPU, имеющий различные области памяти, составляет основу оборудования системы для обработки программ пользователя.

Программируемый контроллер SIMATIC S7-300/400 имеет модульную конструкцию и включает в себя следующие компоненты:

- стойки (Rack), которые используются для размещения в них модулей и для соединения последних друг с другом;
- источник питания (PS - "power supply"), который обеспечивает внутренние напряжения питания;
- центральный процессор (CPU - "central processing unit"), который используется для размещения и обработки программы пользователя;
- интерфейсные модули (IM - "interface module"), которые используются для соединения стоек друг с другом;
- сигнальные модули (SM - "signal module"), которые используются для преобразования сигналов, поступающих от процесса, во внутренние сигналы для последующей переработки в дискретные или аналоговые сигналы для управления приводами.
- функциональные модули (FM - "function module"), который не зависит от CPU и используется для выполнения сложных или зависящих от времени процессов;
- коммуникационные процессоры (CP - "communication processor"), которые используются для связи с подсетями.

Подсети используются для связи программируемых контроллеров друг с другом или с другими устройствами.

Программируемый контроллер (или станция) может состоять из нескольких стоек, которые связываются друг с другом посредством шины. Источник питания, CPU и I/O модули (модули SM, FM и CP) включаются в центральную стойку. Если для I/O модулей недостаточно места или необходимо часть или все I/O модули разместить вне центральной стойки, то в таких случаях используют дополнительные стойки - стойки расширения, которые соединяются с центральной стойкой посредством интерфейсных модулей (см. рис. 1.6).

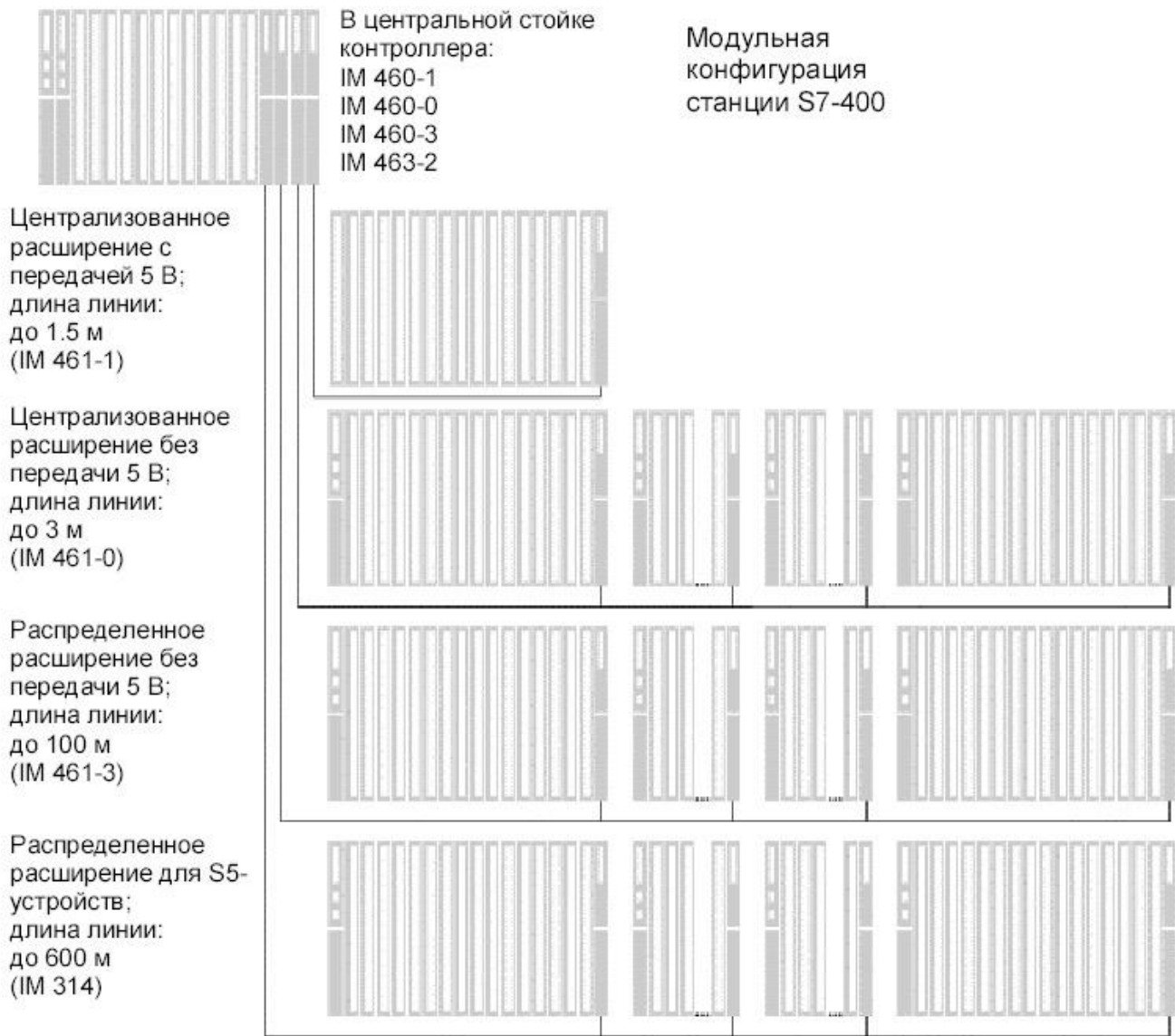


Рис. 1.6

1.5.2. Станция S7-300

Программируемый контроллер S7-300 позволяет включить в центральную монтажную стойку до 8 входных/выходных модулей. Если такая однорядная конфигурация контроллера не является достаточной, то возможны два варианта расширения конфигурации при использовании CPU 314 или более мощных процессоров: вариант двухрядной конфигурации, имеющей центральную стойку и одну стойку расширения (при использовании интерфейсных модулей IM 365 и с расстоянием до одного метра между стойками), или вариант конфигурации, состоящей максимально из 4 рядов, т.е. - центральной стойки, имеющей до 3 стоек расширения (при использовании интерфейсных модулей IM 360 и IM 361 и с

расстоянием до десяти метров между стойками). Вы можете задействовать максимум восемь модулей в стойке. Число модулей может быть ограничено также максимально допустимым током потребления на одну стойку, который составляет 1.2 А (для CPU 312 IFM максимально допустимый ток потребления составляет 0.8 А). Модули связаны между собой внутренней шиной стойки, обеспечивающей функции Р- и К-шин.

Особую возможность при конфигурировании предоставляет использование прикладного модуля FM 356 из семейства компьютеров для автоматизации M7-300. Модуль FM-356 позволяет "разбить" интерфейсную шину модулей контроллера, чтобы получить контроль над оставшимися в "отсеченном сегменте шины" модулями для автономного управления ими. В данном случае ограничивающим фактором также являются такие параметры, как число модулей и суммарная потребляемая ими мощность.

Модули SIMATIC S7-300 допускают использование в жестких внешних условиях. Они имеют расширенный температурный рабочий диапазон от -25 до $+60^{\circ}\text{C}$, повышенную вибрационную и ударную стойкость, соответствующие стандарту IEC 68 часть 2-6; удовлетворяют требованиям по влагостойкости, устойчивости к образованию конденсата и инея согласно IEC 721-3-3 Class 3 K5, также как и требованиям стандарта для ж/д транспорта по EN 50155. Остальные характеристики стандартны.

1.5.3. Станция S7-400

Программируемый контроллер S7-400 имеет следующий состав: центральная монтажная стойка на 18 или 9 слотов (соответственно UR1 или UR2), модуль блока питания и модуль CPU, которые могут сами занимать от одного до нескольких слотов (посадочных мест в монтажной стойке). Интерфейсные модули IM 460-1 и IM 461-1 позволяют использовать одну стойку расширения с подачей в неё от центральной стойки 5-вольтового питающего напряжения с длиной линии до 1.5 метров между стойками. Кроме того, с помощью интерфейсных модулей IM 460-0 и IM 461-0 могут быть использованы от одной до 4 стоек расширения с длиной шины связи с центральной монтажной стойкой до 3 метров. И, наконец, с помощью интерфейсных модулей IM 460-3 и IM 461-3 могут быть использованы от одной до 4 стоек расширения с длиной шины связи с центральной монтажной стойкой до 100 метров.

Максимальное количество подсоединяемых к центральной монтажной стойке стоек расширения составляет 21 единицу. Для распознавания стоек расширения необходимо их количество задавать с помощью кодирующего переключателя на приемном интерфейсном модуле. Внутренняя шина состоит из параллельной Р- и последовательной К-шин. Стойки расширения ER1 и ER2 (соответственно на 18 и 9 слотов) предназначены для "простых" сигнальных модулей, которые не могут генерировать аппаратные прерывания, не имеют 24-вольтового питающего напряжения по Р-шине, не имеют резервного питания и не имеют связи по К-шине. К-шина используется в стойках UR1, UR2 и CR2 или в случае применения их в качестве центральных монтажных стоек, или в случае применения их в качестве монтажных стоек расширения с номерами от 1 до 6.

Особую возможность при конфигурировании предоставляет использование сегментированной монтажной стойки CR2. Сегментированная монтажная стойка CR2 позволяет использовать два центральных процессора с общим для них модулем питания. При этом оба CPU могут сохранять свою функциональную обособленность, так как имеют отдельные Р-шины со своими собственными сигнальными модулями, и могут в то же время обмениваться данными посредством К-шины.

В программируемом контроллере S7-400 возможно организовать мультипроцессорный режим с участием нескольких (до четырех единиц) специальных CPU. При этом в такой станции каждый модуль назначается только одному из CPU (соответственно с его адресацией и прерываниями).

Используя стандартные компоненты SIMATIC S7-300/400, Вы можете создать резервированную на основе программного обеспечения систему с ведущей станцией управления процессом и резервной станцией, которая принимает на себя управление процессом в случае выхода из строя ведущей станции. Устойчивость к сбоям системы управления с резервированием на основе программного обеспечения подходит для так называемых "медленных процессов", так как переключение управления на резервную станцию может потребовать нескольких секунд, в зависимости от конфигурации программируемых контроллеров. Сигналы, поступающие от процесса, "замораживаются" на время перехода управления к резервной станции. Резервная станция будет продолжать работу по управлению процессом, используя последние корректные данные, полученные ведущей станцией.

Резервирование входных/выходных модулей обеспечивается с помощью распределенной периферии (I/O) (ET 200M с интерфейсным модулем IM 153-3 для резервирования PROFIBUS-DP). В системе управления может быть сконфигурировано заказное (опционное) программное обеспечение для резервирования ("Software Redundancy").

SIMATIC S7-400H . это отказоустойчивый программируемый контроллер с резервированной конфигурацией, имеющей две центральные стойки, каждая с H CPU и с модулем синхронизации для сравнения данных с волоконно-оптическим кабелем. Оба регулятора работают в режиме "горячего

резервирования"; в случае отказа неповрежденный контроллер в одиночку продолжает выполнять функции управления после плавного автоматического переключения на резервный режим работы. Входы/выходы могут обеспечивать обычный нормальный доступ (одноканальная, односторонняя конфигурация) или расширенный доступ (одноканальная переключаемая конфигурация с ET 200M). Связь осуществляется по простой или по резервной шине. Программа пользователя точно такая же, как и для не резервированного контроллера; функции резервирования выполняются исключительно аппаратно и не заметны для пользователя. Для конфигурирования системы требуется заказное (опционное) программное обеспечение "S7-400H".

1.5.4. Области памяти CPU

Память пользователя - области памяти CPU, имеющие значение для Вашей программы – загрузочная, рабочая и системная. Программа пользователя собственно располагается в двух областях: в загрузочной памяти (load memory) и в рабочей памяти (work memory). Загрузочная память целиком содержит пользовательскую программу: части программы, выполняемые в любое заданное время (исполняемый модуль программы), находятся в рабочей памяти, обеспечивающей малое время доступа к данным, что предопределяет высокую скорость обработки программы.

Загрузочная память конструктивно может быть частью CPU или может быть в виде встраиваемого отдельного модуля памяти. Вводимая пользователем программа, включая данные конфигурирования, располагается в загрузочной памяти (load memory) и в оперативной памяти.

Рабочая память конструктивно является частью CPU и представляет собой быструю RAM-память. В оперативной памяти содержатся релевантные части программы пользователя: собственно код программы и данные пользователя. Здесь "релевантность" означает, что в эту память загружается код, описывающий существующие объекты, но это не предполагает обязательность вызова отдельных блоков этого кода для обработки.

Системная память содержит адреса (переменные), к которым Вы обращаетесь в своей программе. Все адреса объединяются в области (адресное пространство), содержащие определенное, зависящее от конкретного CPU, число адресов. Адреса эти могут, например, принадлежать входам, используемым для опроса состояния сигналов от кнопок или конечных переключателей, или выходам, используемым для управления контакторами (реле) или лампами. Системная память CPU содержит следующие адресные области.

- Входы (I) формируют "отображение процесса по входам" дискретных входных модулей.
- Выходы (Q) формируют "отображение процесса по выходам" дискретных выходных модулей.
- Меркеры (M) хранят информацию, доступную из любой точки программы.
- Таймеры (T) хранят информацию, определяющую параметры времени для функций ожидания и мониторинга.
- Счетчики (C) хранят информацию для функции прямого и обратного счета.
- Временные локальные данные (L) используются в качестве динамических промежуточных буферов при обработке блоков. Временные локальные данные располагаются в L-стеке, который динамически занимается и высвобождается CPU при выполнении программы.

Буквы, заключенные в скобки в выше приведенных пунктах перечисления адресных областей, представляют собой аббревиатуры, используемые в мнемониках программы при различных способах задания адресов. Вы можете также назначать символьное имя для каждой переменной и затем использовать его вместо идентификатора адреса.

В системной памяти также содержатся буферы для коммуникационных заданий и системных сообщений (буфер диагностики). Размеры этих буферов данных, также как и размеры областей хранения отображения процесса по входу и выходу, в новых центральных процессорах для S7-400 может определять пользователь.

Распределенные входы/выходы (I/O) относятся к модулям, подключенным посредством PROFIBUS-DP к ведущему модулю PROFIBUS. PROFIBUS-DP совместим со стандартом EN 50170 и является независимым от производителей стандартом для подключения стандартных ведомых DP-устройств.

Если Вы загружаете новый или измененный блок в CPU, CPU помещает блок в загрузочную память и передает релевантные данные в рабочую память. Если в рабочей памяти уже находится блок с таким же номером, то этот "старый блок" объявляется неправильным (invalid) (пользователю предлагается это подтвердить), и после этого новый блок добавляется в "конец" занятой области памяти (после последней записи).

Если даже блок удален, он помечается как неправильный (invalid), но в действительности из памяти не удаляется. Это приводит к разрывам (gap) в непрерывном занятом пространстве памяти и уменьшает общий объем доступной памяти. Эти разрывы могут быть заполнены только при использовании

функции **сжатия** *Compress*. При использовании этой функции в режиме RUN блоки, выполняющиеся в текущий момент, не могут быть перемещены; только в режиме STOP Вы можете эффективно ликвидировать разрывы в непрерывном занятом пространстве памяти.

Текущее состояние загрузки памяти может быть отображено в процентах с помощью опций меню: *PLC -> Module Information (PLC -> Информация о модуле)* на вкладке "Memory" ("Память"). В появившемся диалоговом окне Вы можете включить функцию сжатия. С помощью SFC 25 (COMPRESS) Вы можете запрограммировать запуск функции сжатия, управляемый событием.

1.5.5. Коммуникации (Communications)

Коммуникации с помощью сети обеспечивают обмен данными между программируемыми модулями, это встроенный компонент SIMATIC S7. Почти все коммуникационные функции управляются операционной системой. Обмен данными может быть организован без какого-либо дополнительного оборудования посредством только соединительного кабеля между CPU. При использовании модулей CP можно создавать мощные сети и с легкостью подключать к ним системы сторонних (кроме SIEMENS) производителей оборудования.

SIMATIC NET - более широкое понятие, включающее в себя понятие коммуникаций SIMATIC. SIMATIC NET представляет собой информационный обмен между программируемыми контроллерами, а также между программируемыми контроллерами и устройствами HMI (человеко-машинный интерфейс). В SIMATIC могут быть реализованы различные варианты функций связи в зависимости от поставленной задачи.

На рисунке 1.8 показаны наиболее важные объекты связи. Может возникнуть задача реализации обмена данными между станциями SIMATIC или аппаратами сторонних (отличных от SIEMENS) производителей. В этом случае необходимы модули с функцией связи. С помощью SIMATIC S7 все CPU обеспечиваются MPI интерфейсом, с помощью которого они могут связываться друг с другом.

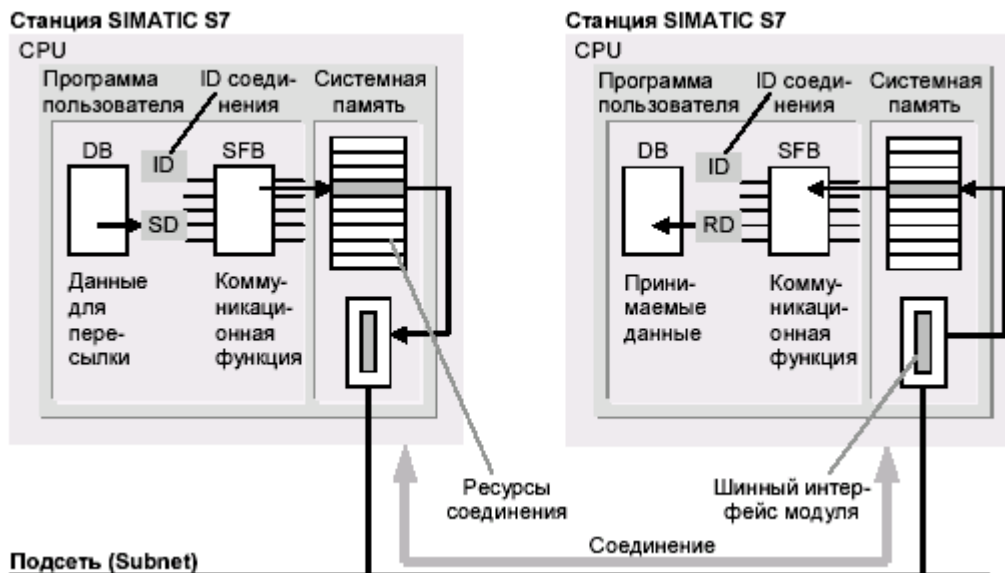


Рис. 1.8.

Кроме того, для связи могут быть применены коммуникационные процессоры (CP), выполняющие обмен данными с высокой пропускной способностью и с различными протоколами обмена.

Модули могут быть связаны сетью. **Сеть** - это соединение между несколькими устройствами с целью их связи друг с другом. Она состоит из одной или нескольких идентичных или разных подсетей, связанных друг с другом. Сеть - это аппаратное соединение между узлами связи (коммуникационными узлами). Обмен данными происходит посредством "соединения" в соответствии со специальным планом обработки данных ("служба обмена"), который основывается на специальной процедуре ("протокол"). Например, S7-соединение является стандартом для S7-модулей с функциями связи.

В **подсети** все коммуникационные узлы связаны с помощью аппаратных соединений, обладающих одинаковыми физическими характеристиками и параметрами передачи, такими как скорость передачи; кроме того, обмен данными в подсети происходит в соответствии с единой процедурой передачи данных. В системе SIMATIC применяются несколько типов подсетей: MPI, PROFIBUS, Industrial Ethernet и PTP ("point-to-point" [соединение "точка к точке"]). Подсети отличаются своими рабочими характеристиками:

- MPI - экономичный способ создания сетей для небольшого количества устройств SIMATIC с обменом малыми количествами данных.
- PROFIBUS - высокоскоростной обмен малыми и средними объемами данных; используется прежде всего для работы с системами распределенных входов/выходов.
- Industrial Ethernet - связь между компьютерами и PLC для высокоскоростного обмена большими объемами данных.
- PTP ("Точка к точке") - последовательная связь между двумя коммуникационными партнерами по специальным протоколам.

Служба обмена (communications service) определяет, как происходит обмен данными между коммуникационными узлами, и как эти данные обрабатываются. Служба обмена базируется на протоколе обмена, который помимо всего прочего описывает процедуру координации работы между коммуникационными узлами.

В SIMATIC различают следующие разновидности организации службы обмена: через функции S7, PROFIBUS-DP, PROFIBUS-FMS, PROFIBUS-FDL (SDA), ISO transport, ISO-on-TSP и связь через глобальные данные.

Соединение (connection) определяет коммуникационные отношения между двумя узлами связи (коммуникационными узлами). Соединение - это логическое назначение двух узлов для выполнения специфических коммуникационных функций (службы обмена) и, кроме того, оно содержит специальные характеристики, такие как тип соединения (динамический, статический) и каким образом оно устанавливается.

В SIMATIC различают следующие разновидности соединений: S7-соединение, S7-соединение (отказоустойчивое), "point-to-point" (соединение "точка к точке"), FMS- и FDL-соединение, "ISO transport"-соединение, "ISO-on-TSP"- и TSP-соединение, UDP-соединение и E-mail-соединение.

Коммуникационные функции (communications functions) играют роль интерфейса между программой пользователя и службой обмена подсети. Используемые для внутренних соединений в SIMATIC S7 коммуникационные функции встроены в операционную систему CPU и вызываются с помощью системных блоков. Загружаемые блоки позволяют создавать соединения с устройствами сторонних производителей (кроме Siemens) с помощью коммуникационных процессоров. В таблице 1.2 показано соответствие между подсетями, службами обмена данными и модулями с функцией связи.

Таблица 1.2. Коммуникационные объекты

Подсеть	Модули	Служба обмена данными	Конфигурация, интерфейс
MPI	Все CPU	Связь через глобальные данные (GD)	GD-таблица
		Связь между станциями посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
PROFIBUS	CPU с ведущим DP-устройством	PROFIBUS-DP (ведущее или ведомое устройство)	Конфигурация оборудования, входы/ выходы, вызовы SFC
		Связь внутри станции посредством SFC	Вызовы SFC
	IM 467	PROFIBUS-DP (ведущее или ведомое устройство)	Конфигурация оборудования, входы/ выходы, вызовы SFC
		Связь внутри станции посредством SFC	Вызовы SFC
	CP 342-5 CP 443-5 Extended (расширенный)	PROFIBUS-FDL, PROFIBUS-DP (ведущее или ведомое устройство)	NCM, таблица соединений, SEND / RECEIVE
		Связь внутри станции посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
	CP 343-5 CP 443-5 Basic (базовый)	PROFIBUS-FMS, PROFIBUS-FDL	NCM, FMS-интерфейс, таблица соединений, SEND / RECEIVE
		Связь внутри станции посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
Industrial Ethernet	CP 343-1 CP 443-1	Транспортные протоколы ISO и TCP / IP	NCM, таблица соединений, SEND / RECEIVE
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
	CP 343-1 IT CP 443-1 IT	Транспортные протоколы ISO и TCP / IP IT-связь	NCM, таблица соединений, SEND / RECEIVE
		Связь посредством SFB (при активной S7-400)	Таблица соединений, вызовы SFB

NCM - программное обеспечение для конфигурирования CP; NCM может применяться для PROFIBUS и для Industrial Ethernet.

Посредством STEP 7 V.5 Вы можете использовать программатор для получения доступа к станциям SIMATIC S7 с помощью подсетей, чтобы, например, задать параметры или изменить программу. Шлюзы (переходы) между подсетями должны быть расположены в станции S7 с возможностью программирования.

Сеть PROFIBUS-DP обеспечивает стандартный интерфейс для передачи преимущественно двоичных данных процесса между "интерфейсным модулем" в центральном программируемом контроллере и приборами полевого уровня. Этот "интерфейсный модуль" называется "ведущим DP-устройством" (DP-master), а приборы полевого уровня называются "ведомыми DP-устройствами" (DP-slave).

Ведущее DP-устройство и все управляемые им ведомые DP-устройства образуют "систему ведущего DP-устройства" (DP-master system). В одном сегменте сети может быть до 32 станций и максимально до 127 станций может быть в сети всего. Ведущему DP-устройству соответствует определенное число, управляемых им ведомых DP-устройств. Пользователь может также подключать к сети PROFIBUS-DP программатор, также как и, например, устройства для обеспечения человеко-машинного интерфейса, устройства ET 200 или ведомые DP-устройства SIMATIC S5.

AS-интерфейс ("AS-Interface", AS-i) позволяет создать подсеть для обмена данными в соответствии со спецификацией IEC TG 178 для AS-интерфейса с соответствующим образом сконструированными двоичными датчиками и приводами. AS-интерфейс не рассматривается как подсеть в SIMATIC Manager; только ведущее устройство AS-i (AS-I master) может быть сконфигурировано процедурами конфигурирования аппаратной части и сети.

1.5.6. MPI

Подсеть "multipoint interface" (MPI) разработана для использования на уровне подразделений предприятия ("cell level"). MPI - многоточечный интерфейс в SIMATIC® S7 и C7. MPI - это, в основном, интерфейс с PG, то есть он разрабатывался для соединения с PG (для запуска и тестирования) и панелями оператора OP (HMI-интерфейс). Подсеть MPI может, тем не менее, использоваться также для сети с небольшим количеством CPU.

Каждый CPU имеет интерфейс для подключения MPI. Он позволяет создать подсеть для обмена данными между CPU, PG, устройствами HMI (человеко-машинный интерфейс) согласно оригинальному протоколу обмена Siemens.

Каждый программатор имеет MPI интерфейс. Интерфейс MPI в CPU позволяет реализовать доступ ко всем интеллектуальным модулям в PLC, в частности к функциональным модулям станции. Каждому узлу MPI-сети нужен собственный MPI-адрес (число от 0 до 32), установки по умолчанию: PG - 0, OP/TD - 1 и CPU - 2. В S7-300™ шина MPI непосредственно, без какого-либо преобразования (1:1) соединена с K шиной. Это означает, что каждый узел на K-шине (модули FM или CP) в стойке S7-300™ является также узлом MPI и должен иметь собственный MPI-адрес. В S7-400™ коммуникационные фреймы MPI (187.5 Кбит/с) преобразуются для передачи по внутренней K-шине (10.5 Мбит/с). В стойке S7-400™ только CPU имеет собственный адрес MPI. Другие интеллектуальные модули например, FM или CP, не имеют своего MPI-адреса.

Для подключения PG или ПК к MPI-интерфейсу PLC требуется:

- MPI модуль в PG / ПК и соединительный кабель,
- **адаптер для ПК (соединительный кабель со встроенным MPI-конвертором, если нет свободного слота в PG / ПК).**

Характеристики адаптера для ПК:

- длина - 5 м;
- скорость передачи до адаптера - 187,5 Кбит/с;
- **скорость передачи от адаптера до PG - 19,2 Кбит/с / 38,4 Кбит/с (настраивается).**

Линии передачи MPI могут иметь два типа исполнения: экранированный кабель "витая пара" или пластмассовый оптико-волоконный кабель. Длина кабеля в шинном сегменте может достигать 50 м. При этом максимальная длина может быть увеличена до 1100 м в случае применения повторителей RS485 или может даже превышать 100 км в случае применения модулей оптической связи (optical link modul). Скорость передачи данных обычно составляет 187,5 кбит/с.

Максимальное число узлов составляет 32 единицы. Каждый узел имеет доступ к шине в течение определенного отрезка времени и может в это время посылать фреймы данных. По прошествии этого промежутка времени узел передает право доступа к шине следующему узлу (процедура доступа "token passing" [передача маркера или "токена"]).

Посредством сети MPI может быть организован обмен данными между CPU с помощью установления одного из следующих типов связи: связи через глобальные данные, связи между станциями посредством SFC или связи посредством SFB. При этом не требуется применять дополнительные модули.

Связь через глобальные данные (Global data communications) позволяет осуществлять обмен небольшими объемами данных между несколькими CPU без дополнительного усложнения программы пользователя. Передача данных может выполняться циклически или запускаться событиями.

Связь через глобальные данные как процедура носит характер "вещания" (распространения данных); получение данных не квитируется. Состояние соединения подтверждается. Связь через глобальные данные возможна только с MPI-шиной и K-шиной.

Связь посредством глобальных данных, не программируется, а конфигурируется. Конфигурация коммуникаций посредством GD-данных хранится в специальной таблице.

Связь посредством GD-данных может быть установлена для нескольких (до 15) CPU внутри проекта. Этот вид связи предназначен для передачи небольших объемов данных, обычно передаваемых с определенной периодичностью.

CPU S7-400™ допускают также программируемую и, следовательно, управляемую событиями пересылку данных.

Конфигурирование связи посредством глобальных данных выполняется с помощью утилиты "Defining Global Data" ("Назначение глобальных данных").

Глобальные данные, то есть переменные, называемые "sender/receiver" (передатчик/получатель), могут собираться в GD-пакете и посылаться вместе. Каждый GD-пакет имеет свой номер, а переменные внутри пакета идентифицируются номерами переменных.

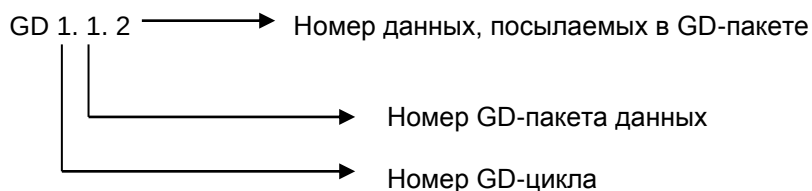
CPU, участвующие в обмене GD-пакетов, образуют GD-цикл. Каждый GD-цикл имеет свой номер. GD-цикл - это список распределения GD-пакетов. Каждый CPU в цикле глобальных данных может посылать данные другим CPU или принимать данные от других CPU.

Возможны следующие **типы GD-циклов**:

- GD-цикл с более, чем с двумя CPU. В этом случае один CPU является передатчиком пакета данных, а другие CPU этого GD- цикла являются приемниками данных.
- GD-цикл с двумя CPU. В этом случае каждый из двух CPU этого GD-цикла является и передатчиком, и приемником GD-пакетов.

Каждый CPU системы S7-300™ может участвовать максимум в 4-х различных GD-циклах. В одной MPI-сети участниками связи посредством глобальных данных могут быть до 15 CPU.

Ниже приведен пример идентификации данных, используемый при обмене данными в GD-цикле.



Один CPU S7-300™ может участвовать максимально в 4-х GD-циклах, может передавать и получать в каждом GD-цикле максимально 1 пакет. В одном пакете может быть передано до 22 байтов данных.

Один CPU S7-400™ может участвовать максимально в 16 GD-циклах, и может в каждом GD-цикле передавать максимально 1 и получать максимально 2 пакета. В одном пакете может быть передано до 54 байтов данных.

Для обеспечения работы MPI-сети необходимо выполнить конфигурирование HW-станций. Прежде всего Вы должны создать станции в проекте для вашей сети, используя SIMATIC® Manager. Когда Вы сделаете это, откройте утилиту "HW Config" и сконфигурируйте последовательно все станции друг за другом.

Чтобы сконфигурировать аппаратные средства для GD-коммуникаций, Вы должны выполнить следующие шаги.

1. В SIMATIC Manager должен был создан STEP 7 -проект.
2. В этом проекте должен быть создан объект "MPI-сеть" с назначенными параметрами. Объект "MPI-сеть" всегда автоматически создается при создании нового S7-проекта.
3. Сконфигурируйте в проекте модули, способные к GD-коммуникациям (например, S7 CPU). Определяя параметры для CPU с помощью утилиты "HW Config", отметьте для каждого CPU параметр "Networked" и назначьте им уникальные MPI-адреса.
4. Загрузите данные конфигурации, которые Вы ввели, в каждый CPU отдельно.
5. Физически соедините все CPU сетевыми кабелями.
6. Используйте в SIMATIC® Manager функцию "Accessible Nodes" ("Доступные узлы"), чтобы проверить правильность включения станций в сеть.

Для обмена через глобальные данные (GD) Вы должны определить коммуникационных партнеров в таблице GD. В таблице глобальных данных введите все CPU, которые должны обмениваться глобальными данными, и области адресов данных, которые будут участвовать в обмене.

Вы должны использовать утилиту "Defining Global Data" ("Определение глобальных данных") для создания таблицы глобальных данных, в которой Вы определяете данные для обмена. Затем Вы дважды компилируете таблицу и загружаете данные конфигурации соединений в CPU.

Вы можете также задать частоту передачи (Scan Rates) и двойное слово для информации о статусе передачи (GD Status).

Откройте GD-таблицу следующим образом:

- 1) откройте свой проект и выберите сетевой объект "MPI";
- 2) выберите опции меню: Options -> Define Global Data (Определение глобальных данных), затем генерируется новая GD-таблица или открывается существующая GD-таблица.

Для заполнения GD таблицы. Вы должны ввести в отдельных столбцах адреса областей данных для каждого CPU, принимающего участие в GD-связи. Вы делаете это следующим образом.

1. Сначала назначьте для каждого столбца CPU, щелкнув кнопкой манипулятора "мышь" на заголовке столбца и затем выбирая пункт меню Edit -> Assign CPU (Правка -> Назначить CPU).
2. Выберите требуемый CPU в диалоговом меню "Select CPU" ("Выбор CPU") и подтвердите "OK".
3. Введите глобальные данные, которые должны передаваться, в ниже лежащих строках (за исключением внешних периферийных входов и выходов и временных данных). Вы можете выбрать режим редактирования (Edit) для отдельных ячеек таблицы с помощью клавиши F2. Вы можете ввести коэффициент повторения для переменных, чтобы определить передачу целого раздела данных, например, 20 байтов, начиная с DBB0 в блоке DB100.
4. Определите передаваемые данные в каждой строке GD-таблицы, выбирая соответствующую ячейку и нажимая на кнопку "Select as Sender" ("Выбрать источник") в панели инструментов.

1.6. Адресация модулей

При монтаже установки нужно трассировать сигналы в PLC (см. рис. 1.9). Входной сигнал, например, сигнал от кнопки +HP01-S10, включающей мотор ("Switch motor on" - "включение мотора"), приходит во входной модуль, где он поступает на специальный терминал. Этот терминал имеет "адрес", называемый I/O-адрес (например, байт 5, бит 2).

Перед каждым началом выполнения программы CPU автоматически копирует значение сигнала в таблицу входов для сохранения "образа процесса по входу", где это значение будет доступно по входному ("input") адресу для дальнейшей обработки (например, I 5.2). При этом выражение "I 5.2" является абсолютным адресом. Вы можете задать в таблице символов имя этому входу в виде алфавитно-цифрового символьного имени, соответствующего абсолютному адресу входного сигнала (такое, как "Switch motor on"). При этом выражение "Switch motor on" является символьным адресом.

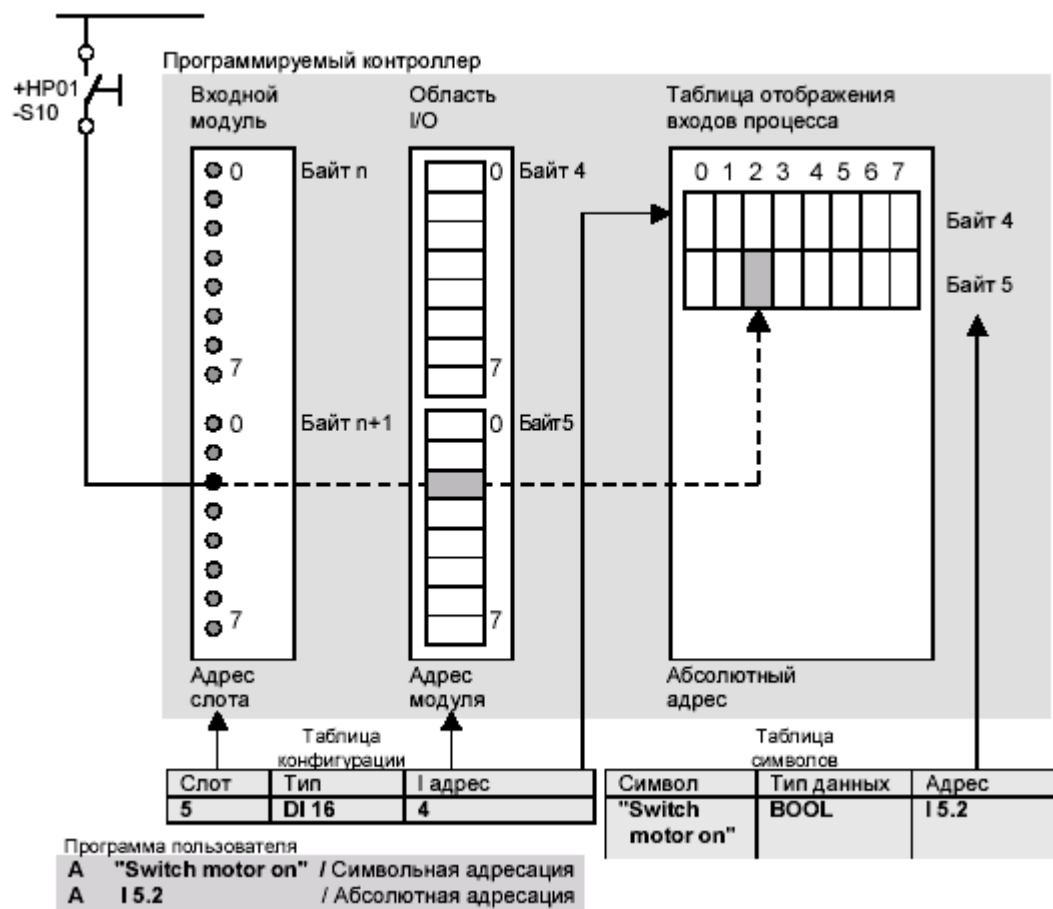


Рис. 1.9. Адресация каналов в модуле.

1.6.1. Адрес слота

Каждый слот имеет фиксированный адрес в программируемом контроллере (в S7-станции). Этот адрес слота состоит из номера монтажной стойки и номера слота. Таким образом каждый модуль может быть однозначно описан указанием адреса слота ("географический адрес"). Если модуль содержит интерфейсные платы, каждая из них также описывается заданием адреса подмодуля. Таким образом, каждый дискретный или аналоговый сигнал и каждое последовательное соединение в системе имеет свой собственный уникальный адрес.

Соответственно, распределенные I/O модули также имеют "географические адреса", в данном случае включающие в себя номер системы ведущего DP-устройства и номер станции вместо номера стойки.

Вы должны использовать утилиту для конфигурирования оборудования "Hardware Configuration" для того, чтобы спланировать конфигурацию аппаратной части S7-станции, как места физического расположения модулей. Эта утилита позволяет также установить начальные адреса модулей и задать для модулей параметры (см. раздел "Конфигурирование станций").

1.6.2. Начальный адрес модуля

Кроме адресов слотов, позволяющих определить отдельный слот, каждый модуль имеет начальный адрес, определяющий позицию в области логических адресов (область I/O-адресов). Адресное пространство входов/выходов начинается с адреса 0 и заканчивается некоторым значением, соответствующим верхней границе, которая определяется типом CPU.

Начальный адрес определяет, как адресуются входные/выходные сигналы в программе программируемого контроллера (S7-станции). В случае дискретных модулей отдельные сигналы (биты) собираются в группы по 8 (т.е. в байты). Эти байты имеют соответствующие адреса 0, 1, 2 и 3; адресация байтов начинается с начального адреса модуля. Например, в дискретном модуле с

четырьмя байтами и с начальным адресом модуля 8 отдельные байты имеют адреса 8, 9, 10 и 11 соответственно.

В случае аналоговых модулей каждый из аналоговых сигналов (в виде напряжения или тока), называемых "каналами" ("channel"), занимает 2 байта. Аналоговые модули, в зависимости от конструкции имеющие 2, 4, 8 и 16 каналов, соответственно занимают 4, 8, 16 или 32 байта адресного пространства.

При включении питания (если не было предустановок) CPU устанавливает начальный адрес каждого модуля, зависящий от типа модуля, от слота и стойки. Этот начальный адрес соответствует начальному байту модуля. Вы можете видеть этот адрес в таблице конфигурации. В системах S7-3xx с интегрированным DP-интерфейсом, S7-318 и S7-400 Вы можете изменять этот адрес. Для этого Вы должны выбрать опцию назначения начальных адресов модулей внутри разрешенного адресного пространства. Вы также можете выбрать опцию для назначения разных начальных адресов для входов и выходов для смешанных дискретных или аналоговых модулей.

Как и централизованные модули, модули распределенной периферии (станции) резервируют соответствующие номера байтов в области I/O-адресов. При этом адреса централизованных модулей и распределенных I/O не должны перекрываться.

Соответствующим образом построенные ведомые DP-устройства могут быть параметризованы таким образом, что особые номера байтов обеспечивают консистентность (логическую связанность) данных при их пересылке. Этим ведомым DP-устройствам соответствует I/O адрес одного байта, которым они адресуются при использовании системных функций SFC 14 (DPRD_DAT) и SFC 15 (DPWR_DAT). Дискретные модули обычно адресуются в таблицах отображения процесса таким образом, что состояния их сигналов могут автоматически обновляться и к ним обеспечивается доступ в области адресов "Input" ("Входы") и "Output" ("Выходы"). Аналоговые модули, FM и CP получают адреса не из области отображения процесса.

1.6.3. Диагностические адреса

Модули со встроенной функцией диагностики обеспечивают пользователя диагностическими данными, которые могут оцениваться в пользовательской программе. Если централизованные модули имеют адрес данных пользователя (начальный адрес модуля), то для чтения диагностических данных имеется доступ к модулю по этому адресу. Для модулей, не имеющих адреса данных пользователя, например, для источников питания, или для модулей, являющихся частью распределенной периферии, существует диагностический адрес.

Адрес данных диагностики всегда является адресом в I/O области и занимает один байт памяти. Длина данных пользователя по этому адресу равна 0; если данные размещены в области отображения процесса (что разрешено), то эти данные игнорируются CPU при обновлении образа процесса.

STEP 7 автоматически назначает диагностический адрес, отсчитывая длину данных вниз от верхнего максимального значения для адресов I/O. Вы можете изменять этот адрес с помощью утилиты для конфигурирования аппаратной части.

Данные диагностики могут быть только считаны с помощью специальных системных функций; попытки доступа по адресу этих данных с помощью операторов загрузки не будут иметь эффекта.

1.6.4. Адреса шинных узлов

Каждая DP-станция (например, ведущее или ведомое DP-устройство или программатор) в подсети PROFIBUS имеет дополнительный адрес узла, с помощью которого станция может быть однозначно адресована на данной шине.

Модули, являющиеся узлами в MPI-сети (например, CPU, FM или CP), также имеют MPI-адрес. Этот адрес играет решающую роль для связи с PG, HMI-устройствами и для обмена посредством глобальных данных.

Нужно заметить, что в старших версиях S7-300 модули FM и CP, работающие в таких станциях, получают MPI-адрес, который выводится из MPI-адреса CPU. В случае CPU 318 модули с MPI-связью локализуются в их собственном сегменте, так как они не имеют MPI-адреса. Они могут быть адресованы программатором посредством номера стойки и номера слота.

1.6.5. Адресное пространство

Адресное пространство каждого программируемого контроллера включает в себя:

- периферийные входы и выходы;
- области отображения процесса по входу и по выходу;
- области меркеров;
- области таймеров и счетчиков (см. главу "Функции таймеров" и главу "Функции счетчиков");
- области L-стека (см раздел "Временные локальные данные").

К перечисленному нужно добавить области кода и блоков данных с локальными (внутри блока) переменными, в зависимости от программы пользователя.

В SIMATIC S7 каждый модуль может иметь две области адресов: область данных пользователя, которая может быть непосредственно адресована с помощью команд L и T и область системных данных для записей данных передачи. При адресации модулей нет разницы в том, локализованы ли модули в стойках при централизованной конфигурации, или используются как распределенные I/O. Все модули находятся в одном и том же (логическом) адресном пространстве.

Адресную область периферийных входов Вы можете использовать при чтении из области данных входных модулей. Часть PI-области адресов соответствует области отображения данных процесса. Эта часть всегда начинается с 0-го адреса I/O, при этом размер этой области определяется типом CPU. С помощью операции прямого чтения (Direct I/O Read) Вы можете получить доступ к данным модулей, чей интерфейс не влияет на образ процесса по входу (например, аналоговые входные модули). Состояния сигналов модулей, которые влияют на образ процесса по входу, могут также быть считаны с помощью этой операции. При этом сканируются мгновенные состояния сигналов входных битов. Необходимо учитывать, что состояния этих сигналов могут отличаться от состояний соответствующих битов области отображения входов процесса, так как образ процесса обновляется в самом начале цикла сканирования программы.

Периферийные входы могут иметь такие же абсолютные адреса, как и периферийные выходы. Меркеры могут рассматриваться как дополнительные "пусковые реле" контроллера. Меркеры используются в первую очередь для хранения состояния сигналов. Они могут трактоваться как виртуальные выходы. Меркеры располагаются в области системной памяти CPU, и, следовательно, они всегда доступны. Имеющееся количество меркеров зависит от типа выбранного CPU. Меркеры используются для хранения промежуточных результатов, которые действительны за пределами блока, и могут обрабатываться более чем в одном блоке. Кроме блоков глобальных данных для хранения промежуточных результатов подходят:

- Временные локальные данные, возможные во всех блоках, но являющиеся действительными только для текущего вызова блока.
- Статические локальные данные, возможные только в функциональных блоках, но являющиеся действительными для многих вызовов блоков.

Некоторые меркеры могут быть назначены **реманентными меркерами**, что означает, что эти меркеры сохраняют свое состояние даже в условиях выключения питания. Реманентная область всегда начинается с 0-го адреса и заканчивается в заданном месте памяти. Реманентная область может быть задана при параметризации CPU.

Многие процедуры в контроллере требуют периодического сигнала. Такие сигналы могут быть получены с помощью таймеров (генератор тактовых импульсов), таймерных (watchdog) прерываний (выполнение программы с управлением по времени) или просто с использованием тактовых меркеров.

Тактовые меркеры - это биты, состояния сигнала которых меняются периодически с отношением сигнал/пауза, равным 1:1. Такие биты, объединенные в байт, обеспечивают фиксированные частоты периодических колебаний (см. рис.1.10).

Вы можете задать число тактовых меркеров при параметризации CPU. Необходимо отметить, что обновление тактовых меркеров асинхронно по отношению к выполнению главной программы.

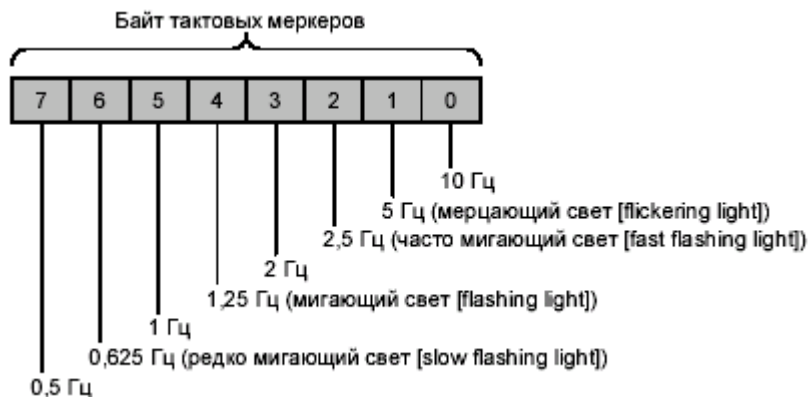


Рис. 1.10

2. Программное обеспечение STEP 7

Базовый пакет STEP 7 (STEP 7 Basic Package) содержит следующие языки программирования: STL ("statement list" - список мнемоник), LAD ("ladder diagram" - контактный план), FBD ("function block diagram" - функциональный план). В дополнение к базовому пакету возможна поставка по специальному заказу пакетов S7-SCL ("Structured Control Language"), S7-GRAH и другие.

Пакет STEP 7 V 5 является набором 32-разрядных приложений, работающих под управлением операционных систем Microsoft Windows 98 или Windows NT (или более поздних).

Пакет STEP 7 V 5 требует от 200 до 380 Мб памяти на жестком диске для каждой локализации (например, для англоязычной) в зависимости от операционной системы и используемой файловой системы. Также требуется предусмотреть свободное место на жестком диске для файла подкачки (от 128 до 256 Мб).

Для инсталляции STEP 7 служит программа SETUP для Windows 9x/NT, которую Вы можете найти на компакт-диске. На программаторах (PG) STEP 7 устанавливается производителем. Кроме STEP 7 компакт-диск также содержит программу авторизации (см. ниже), программу NCM для конфигурирования коммуникационных процессоров и электронные справочники по STEP 7 в формате Acrobat Reader V3.01.

Для интерактивного подключения к контроллеру требуется MPI-интерфейс. Программаторы имеют встроенный MPI-интерфейс, тогда как компьютеры требуют установки MPI-модуля или соединения с помощью PC-адаптера.

Для работы с пакетом STEP 7 требуется выполнить **авторизацию** (подтвердить право использования). Ключи авторизации находятся на дискете. После инсталляции STEP 7 Вам будет предложено выполнить авторизацию, если данный жесткий диск пока не содержит авторизации. Вы также можете выполнить авторизацию позже, спустя некоторое время. Вы можете также переносить авторизацию на другой ПК, сначала возвратив авторизацию на дискету-оригинал и затем установив ее на другом ПК.

В случае потери Вами авторизации, например, из-за выхода из строя жесткого диска, Вы можете в течение ограниченного времени (до замены Вашей авторизации) использовать "аварийную лицензию" (emergency license), находящуюся на той же дискете-оригинале (дискете с авторизацией).

2.1. SIMATIC Manager

SIMATIC Manager является главной утилитой STEP 7. Вы найдете ее иконку на рабочем столе Windows. SIMATIC Manager запускается двойным щелчком кнопкой мыши на иконке.

С помощью SIMATIC Manager Вы сможете работать в среде STEP 7. "Логические" объекты, отображаемые в окнах SIMATIC Manager, соответствуют "реальным" объектам Вашей установки (процесса). Проект включает в себя установку (процесс) в целом, тогда как станция (station) соответствует программируемому контроллеру (PLC).

Проект может содержать несколько станций, связанных друг с другом, например, посредством подсети MPI. Станция содержит CPU, а CPU содержит S7-программу. В свою очередь программа включает в себя другие объекты, такие как контейнер *Blocks* (блоки), содержащий среди прочего скомпилированные блоки.

Объекты STEP 7 объединяются в древовидную структуру. Наиболее важны компоненты этой древовидной структуры ("main branch" - "главная ветвь"), которые характерны для работы с базовым

пакетом S7 в автономном режиме (offline view). Объекты, выделенные жирным шрифтом, содержат другие объекты. В автономном режиме (offline view) все показанные объекты доступны пользователю. Эти объекты расположены на жестком диске программатора. Если Ваш PG находится в интерактивной связи (online) с CPU (обычная система управления с PLC), Вы можете включить интерактивный режим (online view), выбрав опции меню: *View -> Online (Режим -> Интерактивный)*. Эта опция вызывает другое окно проекта, содержащее объекты назначенного устройства.

Вы можете видеть на панели заголовка окна активного проекта, работаете ли Вы в интерактивном (online) или в автономном (offline) режиме. Для более четкого разделения для панели заголовка и заголовка окна этих режимов могут быть установлены различные цвета. Для этого выберите опции меню: *Options -> Customize (Опции -> Установки пользователя)* и измените соответствующие параметры на вкладке "View" ("Режим").

Выбрав опции меню: *Options -> Customize (Опции -> Установки пользователя)*, можно изменить базовые установки SIMATIC Manager, такие как session language (язык), архив программы и место расположения для проектов, библиотек и конфигурирование архива программы.

2.1.1. Проекты и библиотеки

В STEP 7 "главные объекты", находящиеся на верхнем уровне структурной иерархии, это проекты (project) и библиотеки (library). Проекты используются для систематического хранения данных и программ решения задачи автоматизации. Важнейшие из них:

- данные конфигурации оборудования;
- параметры для модулей;
- данные конфигурации сетевых коммуникаций;
- программы (коды и данные, символы, исходные программы).

Объекты в проекте организованы в виде иерархической системы. Первым шагом для редактирования всех объектов проекта является открытие проекта.

Библиотеки используются для хранения многократно используемых компонентов программы. Библиотеки организованы в виде иерархической системы. Они могут содержать STEP 7 программы, которые в свою очередь могут содержать программы пользователя (скомпилированные блоки), исходные тексты программ и таблицы символов. За исключением возможности интерактивной (online) связи (без неё невозможна отладка программы), создание программ или частей программ в библиотеке обеспечивает такие же функциональные возможности как и у объекта. В комплекте поставки STEP 7 V5 находится стандартная библиотека Standard Library, содержащая следующие разделы:

- System Function Blocks (системные функциональные блоки), содержащие интерфейсы вызовов системных блоков для создания программ в автономном режиме, что функционально обеспечивается в CPU;
- S5-S7 Converting Blocks (блоки S5-S7 преобразования), содержащие загружаемые функции для S5-S7 преобразования (для замены стандартных функциональных блоков S5 в процессе конвертирования программы);
- TI-S7 Converting Blocks (блоки TI-S7 преобразования), содержащие дополнительные загружаемые функции и функциональные блоки для TI-S7 преобразования;
- IEC Function Blocks (функциональные блоки IEC), содержащие загружаемые функции для редактирования комплексных переменных типов DATE_AND_TIME и STRING;
- Communication Blocks (коммуникационные блоки), содержащие загружаемые функции для управления модулями CP;
- PID Control Blocks (блоки ПИД-управления), содержащий загружаемые функциональные блоки для систем автоматического управления;
- Organization Blocks (организационные блоки), содержащий шаблоны для организационных блоков (раздел объявления переменных для стартовой информации).

2.1.2. Интерактивная справочная система (Online Help)

Интерактивная справочная система (Online Help) в SIMATIC Manager обеспечит Вас информацией в процессе программирования, что снимает необходимость пользования печатными справочными руководствами. Вы можете выбирать интересующие Вас темы, выбрав пункт меню Help (Справка). Выбор в справочной системе пункта Getting Started (Запуск), к примеру, выводит краткое резюме по использованию утилиты SIMATIC Manager.

При выборе опций: Help -> Contents (Справка -> Содержание) запускается центральная функция справочной системы STEP 7 из любого приложения. Она содержит все основные сведения.

При выборе опций: Help -> Context-Sensitive Help (Справка -> Контекстная справка) либо нажатии F1 запускается контекстная справочная система, то есть если Вы нажмете клавишу F1, то Вы получите информацию, соответствующую выбранному с помощью манипулятора "мышь" объекту, или информацию, соответствующую текущему сообщению об ошибке.

Если на панели Вы щелкните на кнопке со знаками стрелки и вопроса, символ вопроса добавится к указателю мыши. Установив такой указатель на объект (например, на символ или команду меню), Вы получите соответствующую интерактивную справочную информацию.

2.2. Конфигурирование станции

При создании проекта Вы должны создать "контейнеры" ("папки") для данных проекта, затем Вы должны сгенерировать эти данные и занести их в эти контейнеры. Обычно Вы создаете проект, применяя подходящее оборудование, конфигурируете это оборудование (по крайней мере, CPU) и создаете контейнер для программы пользователя. Тем не менее, Вы можете поместить S7-программу непосредственно в контейнер проекта без включения какого-либо оборудования вообще. Заметьте, что инициализация модулей (изменение адресов, установки CPU, конфигурирование соединений) возможна только с помощью утилиты конфигурирования оборудования Hardware Configuration tool. Конфигурирование – альтернатива кодированию (описанию) структуры.

Редактирование проекта в целом необходимо выполнять с использованием SIMATIC Manager.

2.2.1. Создание проекта с S7-станцией

Выберите опции меню: *File -> New (Файл -> Создать)*, введите имя проекта в диалоговом окне, измените тип и место его расположения, если это необходимо, и подтвердите Ваш выбор щелчком на кнопке "ОК" или нажатием клавиши "Enter".

Выберите проект и вставьте станцию с помощью опций меню: *Insert -> Station -> Simatic 300 Station (Вставка -> Станция -> Станция S7-300)* (в данном случае станция S7-300).

Для конфигурирования станции щелкните на прямоугольнике со значком плюса, следующем за объектом *project* в левой части окна проекта и выберите станцию; SIMATIC Manager отображает объект Hardware (оборудование) в правой части окна. Двойным щелчком по *Hardware* запускается утилита конфигурирования оборудования (HW Config), с помощью которой осуществляется создание/редактирование таблиц конфигурации.

При открытии утилиты для конфигурирования оборудования Hardware Configuration отображается окно станции и каталог оборудования.

Для удобства редактирования Вы можете увеличить (максимизировать) окно станции. В верхней части окна показаны монтажные стойки в форме таблиц и DP-станции в форме символов. Если используются несколько стоек, то Вы можете видеть соединения между интерфейсными модулями, а если используется подсеть PROFIBUS, Вы можете видеть систему ведущего DP-устройства. В нижней части окна станции показана таблица конфигурации, которая дает подробную информацию о стойке или о ведомом DP-устройстве, выбранном в верхней части окна.

Если каталог модулей не показан на экране, то вызовите его с помощью опций меню: *View -> Catalog (Вид -> Каталог)*.

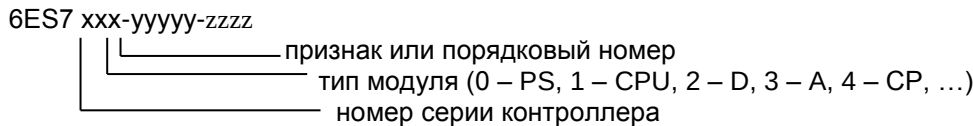
Конфигурирование начинается с выбора несущей шины, например, (rail) в "SIMATIC 300" \ "RACK 300" и переносом методом "drag-n-drop" посредством мыши на свободное место в верхней половине окна станции (station window). При этом Вы можете наблюдать таблицу, в которой показаны слоты на стойке. На следующем этапе Вы должны выбирать требуемые модули из каталога модулей и, используя процедуру "drag-n-drop", переносить эти модули в соответствующие слоты. Для дальнейшего редактирования структуры проекта требуется установить по крайней мере один CPU, например, CPU 314 в слот 2. Вы можете добавлять остальные необходимые модули позже. Затем Вы должны сохранить и скомпилировать станцию, после чего закройте ее и вернитесь в SIMATIC Manager. Кроме конфигурации оборудования открытая станция показывает также CPU. При конфигурировании CPU утилита SIMATIC Manager также создает S7-программу со всеми объектами. Создание структуры проекта при этом завершается.

Вы можете скрывать и показывать каталог оборудования с помощью опций: *View -> Catalog (Вид -> Каталог)*. Каталог отображает все доступные монтажные стойки, модули, интерфейсные модули, совместимые с STEP 7. С помощью опций: *Options -> Edit Catalog Profile (Опции -> Редактирование профиля каталога)* Вы можете скомпилировать свой собственный каталог оборудования, который будет отображать только тот набор модулей, который используется Вами в структурах контроллеров. Двойным щелчком на панели заголовка Вы можете "пристыковать" каталог к правому краю окна станции или вновь освободить его.

2.2.2. Таблица конфигурации (Configuration table)

Утилита для конфигурирования оборудования Hardware Configuration работает с таблицами, каждая из которых представляет монтажную стойку, модуль или DP-станцию. Таблица конфигурации показывает слоты с модулями, установленными в них или свойства модулей, такие как адреса или порядковые номера. Двойной щелчок на строке модуля открывает окно свойств модуля (properties), с помощью этого окна можно задать параметры модуля.

Каждый модуль имеет свой заказной номер, например, 6ES7 972-0BB12-0XA0 (штекер для кабеля). Структура заказного номера имеет вид:



Признак для сигнальных модулей принимает значения: 1 – вход, 2 – выход, 3 – вход/выход.

Символ, говорящий о невозможности назначения данного слота "No Parking" ("Нет установки") для выбранного модуля, появится при попытке назначения уже назначенного слота.

В случае однорядной S7-300 станции слот 3 оставляется пустым: он резервируется для интерфейсного модуля связи со стойкой расширения. Вы можете сгенерировать таблицу конфигурации для другой стойки переносом с помощью мыши выбранной монтажной стойки из каталога в окно станции. В системах S7-400 несоединенной стойке (более точно: соответствующему приемному интерфейсному модулю) назначается интерфейс с помощью вкладки "Link" (соединение) в окне свойств ("Properties") передающего ("Send") IM. Для этого выберите модуль, затем опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*.

При конфигурировании модулей утилита конфигурирования оборудования Hardware Configuration автоматически назначает начальные адреса модулей.

2.2.3. Параметризация модулей

При назначении параметров модулям определяются их свойства. Задавать параметры модуля необходимо, только когда Вы хотите изменить параметры, заданные по умолчанию. Для параметризации модуля требуется, чтобы он был в таблице конфигурации. Для редактирования свойства модуля открываются или двойным щелчком на модуле в таблице конфигурации, или выбором в таблице строки с модулем с последующим выбором опций меню: *Edit -> Object Properties (Правка -> Свойства объекта)*. В раскрывшемся диалоговом окне Вы найдете несколько вкладок с определяемыми параметрами. При использовании этого метода для параметризации CPU Вы можете задавать характеристики выполнения Вашей программы пользователя.

Некоторые модули позволяют устанавливать их параметры в процессе выполнения программы с помощью программы пользователя посредством системных функций SFC 55 WR_PARM, SFC 56 WR_DPARM и SFC 57 PARM_MOD.

2.3. Создание S7-программ

Программа пользователя создается в каталоге (в объекте) *S7 Program*. Вы можете назначать этот объект в структурной иерархии проекта в объекте CPU, или вне зависимости от CPU. В свою очередь объект *S7 Program* включает в себя объект *Symbols (Символы)* и каталоги *Source Files (Исходные файлы)* и *Blocks (Блоки)*.

В случае создания программы путем написания исходных файлов ("**source-oriented**") Вы должны создать одну или несколько исходных программ и сохранить их в виде файлов в каталоге *Source Files (Исходные файлы)*. Исходные программы - это текстовые файлы формата ASCII, которые содержат операторы программы для одного или нескольких блоков, возможно даже целиком всю программу. Вы должны скомпилировать исходные программы, тогда скомпилированные блоки программы помещаются в каталог *Blocks (Блоки)*. Скомпилированные блоки содержат код MC7 и выполняются в S7 CPU.

В случае создания программы "инкрементным" путем (методом добавления), Вы вводите программу строка за строкой. Вводимые строки немедленно проверяются на наличие синтаксических ошибок. При поступлении команды на сохранение программный блок сначала компилируется, затем

сохраняется в каталоге *Blocks (Блоки)*. При создании программы данным методом Вы можете также редактировать блоки в интерактивном (online) режиме в CPU, даже во время рабочего режима.

В программе обрабатываются значения сигналов или значения адресов. Адрес - это, например, вход I1.0 (абсолютная адресация). С помощью таблицы символов **Symbol Table** (объект Symbols) Вы можете назначить адресу символьное имя, например, "Switch motor on" и после этого обращаться к этому адресу, используя данное символьное имя (символьный адрес). В свойствах автономного контейнера *Blocks* Вы можете определить, каким способом будут адресоваться переменные в таблице символов (Symbol Table) в скомпилированных блоках после корректировки - абсолютным или символьным, согласно приоритету адресации (*address priority*).

Требования к памяти для скомпилированного блока можно найти в свойствах блока, для этого при выбранном в SIMATIC Manager блоке выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, тогда на вкладке "General-Part 2" ("Общие - часть 2") Вы можете узнать требования к памяти для Вашей программы в целом. На вкладке *Blocks (Блоки)* Вы найдете размер программы в загрузочной памяти (load memory) и в рабочей памяти (work memory), так же как и число блоков каждого типа. Системные данные не учитываются для этой информации; они занимают дополнительное пространство в загрузочной памяти.

2.3.1. Таблица символов (Symbol Table)

В управляющей программе Вы имеете дело с адресами, т.е. с входами, с выходами, таймерами и блоками. Вы можете назначать абсолютные адреса (например, I1.0) или символьные адреса (например, Start signal). При символьной адресации используются символьные имена. Это делает программу легко читаемой, благодаря тому, что символьные имена несут смысловую нагрузку.

При использовании символьной адресации различаются локальные (*local*) и глобальные (*global*) символьные имена (символы). Локальный символ распознается только в блоке, в котором он определен. Поэтому при необходимости Вы можете использовать одинаковые локальные символьные имена в различных целях в разных блоках. Глобальный символ распознается в любом месте программы и имеет одинаковое значение во всех блоках программы. Вы должны определить глобальный символ в таблице символов.

Глобальный символ начинается с символа алфавита и может иметь в длину до 24 символов. Глобальный символ может также содержать пробелы, специальные символы и национальные символы, например, такие как умляут. Исключения составляют символы 00_{hex}, FF_{hex} и кавычки (""). При наборе имени Вы должны заключать спецсимволы в кавычки. В скомпилированном блоке программный редактор отображает все глобальные символы (символьные адреса) в кавычках. Комментарий к символу может составлять в свою очередь запись из 80 символов.

В таблице символов Вы можете назначать имена следующим адресам и объектам:

- входам I, выходам Q, периферийным входам PI и выходам PQ,
- меркерам M, таймерам T и счетчикам C,
- блокам кодов OB, FB, FC, SFC, SFB и блокам данных DB,
- типам данных, определенным пользователем, UDT,
- таблице переменных VAT.

Адреса данных в блоках данных находятся среди локальных адресов; в случае глобальных блоков данных связанные символы определяются в разделе описаний (declaration section) блоков данных и в разделе описаний (declaration section) функциональных блоков в случае экземплярных блоков данных.

При создании S7-программ SIMATIC Manager создает также пустую таблицу символов *Symbols*. Вы можете открыть эту таблицу и определить глобальные символы и назначить их абсолютным адресам.

В S7-программе может быть только одна таблица символов. Тип данных является частью определения символа. Он определяет особые свойства данных, в частности, представление содержимого данных. Например, тип данных BOOL идентифицирует двоичную переменную, а тип данных INT обозначает переменную в цифровой форме, содержание которой определяется 16-битным целым числом. Для получения более подробной информации см. раздел "Переменные и константы" и раздел "Типы данных", содержащие соответственно обзор и подробное описание типов данных в STEP 7.

В случае "инкрементного" программирования Вы создаете таблицу символов до ввода программы; Вы можете также добавить или скорректировать отдельные символы во время ввода программы. При создании программы путем, ориентированным на создание исходных текстов программы, готовая таблица символов должна быть доступна к моменту компиляции программы.

Таблица символов может быть импортирована и экспортирована. Здесь "экспортируемый" файл означает созданный файл, содержащий данные Вашей таблицы символов. Здесь может быть как таблица символов целиком, так и отдельные строки таблицы. Для файла Вы можете выбирать следующие форматы данных: текстовый ASCII (с расширением *.asc), sequential assignment list (последовательный

список выражений - с расширением *.seq), System Data Format (формат системных данных - с расширением *.sdf для Microsoft Access) и Data Interchange Format (формат обмена данными - с расширением *.dif для Microsoft Excel). Вы можете редактировать экспортируемый файл с помощью подходящего редактора. Вы можете также импортировать таблицу символов в одном из выше упомянутых форматов.

Выбрав опции меню: *Edit -> Special Object Properties (Правка -> Специальные свойства объекта)*, Вы можете установить атрибуты для каждого символа в таблице символов. Эти атрибуты или свойства используются для следующих целей:

- для HMI функций для мониторинга с использованием WinCC,
- для конфигурирования коммуникаций,
- для конфигурирования сообщений,
- для мониторинга процесса посредством S7-PDIAG.

Выбрав опции меню: *View -> Columns O, M, C, R (Вид -> Столбцы O, M, C, R)*, Вы сделаете атрибуты видимыми. С помощью опций меню: *Options -> Customize (Опции -> Установка пользователя)* Вы можете установить, будут или нет специальные свойства объектов копироваться при импортировании символов.

2.3.2. Редактор STL-программ

Для создания программы пользователя в базовый пакет STEP 7 включены редакторы для языков программирования LAD, FBD и STL. При работе с редактором STL-программы Вы можете вводить программу "инкрементно" (непосредственно) или генерировать исходный текст программы и компилировать его позднее.

STL-блоки могут быть "декомпилированы" ("decompiled"), т.е. из MC7 кода может быть извлечен пригодный для чтения блок без автономной базы данных. (Вы можете прочитать любой блок из CPU, используя программатор). Вдобавок, исходная STL- программа может быть восстановлена из любого скомпилированного блока.

Вы можете получить доступ к редактору программ при открытии блока в SIMATIC Manager, например, двойным щелчком на автоматически сгенерированной иконке для организационного блока OB1 или с помощью меню панели задач Windows: *Start -> Simatic -> STEP 7 -> LAD, STL, FBD - Program S7 Blocks*.

Вы можете задать свойства для редактора программ с помощью опций меню: *Options -> Customize (Опции -> Установка пользователя)*. На вкладке "Editor" ("Редактор") можно выбрать свойства, с которыми новый блок должен быть сгенерирован и отображен, такие, например, как язык создания, установки для комментариев и символов.

При открытии скомпилированного блока в каталоге *Blocks (например, двойным щелчком)* блок открывается для инкрементного программирования. Для программирования, ориентированного на создание исходных текстов программы, Вы должны открывать исходный файл программы в каталоге *Source files (Исходные файлы)*.

Вы можете также создавать программу, используя попеременно то один метод, то другой метод программирования, т.е. некоторые блоки вводятся непосредственно, а другие блоки создаются с помощью исходных файлов. Также в программе можно вызывать отдельные блоки, созданные с использованием других языков программирования, таких как LAD и FBD.

Программа пользователя создается блок за блоком и в результате представляет собой исполняемую программу в коде MC7 независимо от языка программирования. Для создания программы пользователя рекомендуется применять метод, ориентированный на создание исходных текстов программы, с использованием символьной адресации. Редактирование получается проще, меньше случается синтаксических ошибок и можно использовать какой-либо другой текстовый редактор. С помощью таблицы символов Вы можете определять различные абсолютные адреса всякий раз перед компиляцией программы, так, что Вы можете создавать многократно используемые "стандартные программы" независимо от конфигурации оборудования.

Способ программирования, ориентированный на создание исходных текстов программы, является единственно возможным способом, обеспечивающим блокам Вашей программы защиту (block protection KNOW_HOW_PROTECT). Инкрементное программирование, тем не менее, является оптимальным для быстрой проверки изменений в программе непосредственно в CPU. Если изменение программы выдержало проверку, обновите и вновь генерируйте исходную программу. Таким образом, у Вас всегда будет текущая версия программы в формате ASCII текстового файла. Инкрементное программирование также очень удобно для тестирования программы с помощью нескольких операторов, включаемых в интерактивном (online) режиме, которые в дальнейшем (после отладки) использоваться не будут.

При инкрементном программировании Вы можете редактировать блоки и в автономном (offline), и в интерактивном (online) режимах работы редактора. При данном методе программирования редактор проверяет введенные в программу изменения сразу же после завершения текущей строки программы.

При закрытии блока, он немедленно компилируется, так что сохранить можно только блоки, не содержащие ошибок.

На вкладке "Create Block" ("Создать блок"), выбранной с помощью опций меню: *Options -> Customize* (Опции -> Установки пользователя), Вы можете выбрать опцию автоматического обновления ссылок (reference data) при создании блока.

Блоки могут редактироваться в автономном (offline) режиме в базе данных программатора и в интерактивном (online) режиме в CPU (или, как обычно говорится, в программируемом контроллере - в PLC). Для этих целей в SIMATIC Manager используются два окна ("offline" и "online"); окна отличаются друг от друга строкой заголовка. В "автономном" ("offline") окне Вы можете редактировать блоки непосредственно в базе данных PG. Из среды редактора Вы можете сохранить измененный блок в автономной ("offline") базе данных с помощью опций меню: *File -> Save* (Файл -> Сохранить) и передать его в CPU посредством опций меню: *PLC -> Download* (PLC -> Загрузить). Если нужно сохранить открытый блок с другим номером или в другом проекте, если необходимо переслать его в библиотеку или в другой CPU, то Вы должны использовать команду: *File -> Save as* (Файл -> Сохранить как).

2.3.3. Ссылки

В добавление к собственно программе утилита SIMATIC Manager предоставляет Вам ссылки (Reference Data), которые Вы можете использовать как основу для корректировки или тестирования программы. Ссылочные данные содержат перечисленную ниже информацию.

- Cross references (Перекрестные ссылки)
- Assignments (Зарезервированные области: I, Q, M, T, C)
- Program structure (Структура программы)
- Unused symbols (Неиспользуемые символы)
- Addresses without symbols (Адреса без символов)

Для генерации ссылочных данных выберите объект *Blocks* (Блоки) и затем опции меню: *Options -> Reference Data -> Display* (Опции -> Ссылочные данные -> Отобразить). Представление ссылочных данных может изменяться в соответствии с рабочим окном с помощью опций: *View -> Filter*. (Вид -> Фильтр.); Вы можете сохранять установки для последующих сессий редактирования с помощью опций: *Save as Standard* (Сохранить как стандарт). Вы можете выводить на экран одновременно несколько списков.

С помощью опций меню: *Options -> Customize* (Опции -> Установки пользователя) в редакторе программы на вкладке "Create Blocks" ("Создать блоки") Вы можете включить или отменить обновление ссылочных данных при компиляции исходного файла программы или при сохранении инкрементно введенного блока.

Примечание: ссылки доступны только при автономной (offline) обработке данных; "автономные" (offline) ссылочные данные могут отображаться даже, если функция вызывается в блоке, открытом интерактивно (online).

Список **перекрестных ссылок** показывает использование адресов и блоков в программе пользователя. В него входят абсолютные адреса, символьные (если есть), блоки, в которых есть обращение к адресу, информация - как используется адрес (для записи или для чтения) и зависящая от языка программирования информация. Для STL-программы столбец, зависящий от языка программирования, содержит номер фрагмента кода, в котором адрес используется. Щелчок на заголовке столбца служит для сортировки таблицы по содержанию столбца. Если выбран адрес, то с помощью выбора опций меню: *Edit -> Go To -> Line* (Правка -> Перейти -> На строку) можете запустить редактор программ, и при этом будет отображен фрагмент программы с выбранным адресом. Список перекрестных ссылок можно просматривать с использованием фильтра (опции: *View -> Filter*. (Вид -> Фильтр.)) для получения информации по интересующим адресам (например, проверить использование меркеров). Если дважды щелкнуть на адресе, в редакторе будет открыт блок на строке, в которой данный адрес используется. STEP 7 всякий раз при открытии списка перекрестных ссылок использует сохраненную (как "Standard") настройку фильтра.

Польза от применения: перекрестные ссылки показывают, были ли адреса использованы для считывания или сброса сигнала. Также они показывают, в каких блоках адреса использованы (возможно, больше одного раза).

Список **назначения (Assignments)** областей памяти I/Q/M показывает, какие биты в адресных областях I, Q и M имеют назначения в программе. В каждой строке показывается побитно один байт. Также указывается тип доступа (побайтный, пословный или двухсловный). Список ссылок на T/C показывает таймеры и счетчики, использованные в программе. В строке показываются по десять таймеров или счетчиков.

Польза от применения: список ссылок показывает, были ли определенные адресные области назначены (заняты) или какие адреса остаются все еще доступными для дальнейшего использования.

Структура программы (Program structure) показывает иерархию вызовов блоков в пользовательской программе. Блоки вызова ("starting blocks") в иерархии вызовов выбираются с помощью настроек фильтров. Вы можете выбирать из двух различных видов: *tree structure* и *Parent-child structure*

"Древовидная" структура (*tree structure*) показывает все уровни вложения в системе вызовов блоков. Пользователь может изменять отображение уровней вложения, используя элементы управления в структуре - квадраты со значками "+" и "-". Требования, предъявляемые к временным локальным данным ("temporary local data"), показаны в целом для цепочки ("path") вызовов. Если щелкнуть правой кнопкой мыши в поле разворачивающегося меню для открытия соответствующего блока, то перейдете к месту вызова или к экрану с дополнительной информацией по блоку.

Структура "Родитель-потомок" (*Parent-child structure*) показывает 2 уровня с одним вызываемым блоком, а также информацию, зависящую от языка.

Польза от применения: структура программы показывает, какие блоки используются, все ли запрограммированные блоки вызываются, какие требования предъявляются к временным локальным данным, удовлетворяют ли локальные данные определенным требованиям для приоритетного класса (для организационного блока).

Список **неиспользуемых символов (Unused symbols)** показывает все адреса, имеющие назначения в таблице символов, но не используются в программе. Список показывает символ, адрес, тип данных и комментарий из таблицы символов.

Польза от применения: список позволяет определить, не были ли отдельные адреса по невнимательности забыты при записи программы, или, может быть, они избыточны и в действительности не нужны.

Список **адресов без символов (Addresses without symbols)** показывает все используемые в программе адреса, которые не имеют символьных имён. Список показывает сами адреса, и как часто они используются.

Польза от применения: список позволяет определить, не используются ли отдельные адреса по невнимательности, (по случайности или по ошибке).

2.3.4. Режимы Online и offline

Пользователь создает конфигурацию оборудования и пользовательскую программу, в основном опираясь на технические средства (engineering system). S7-программа сохраняется автономно (offline) на жестком диске в текстовой форме, а также в скомпилированном виде. Для переноса программы в CPU Вы должны соединить программатор с CPU, после чего - установить интерактивное ("online") соединение. Вы можете использовать это соединение для определения рабочего состояния CPU и назначенных модулей, т.е. Вы можете реализовать функции диагностики.

Соединение программатора и CPU с помощью MPI-интерфейса является аппаратным требованием для интерактивного (online) соединения. Во всех случаях программатор может быть задействован в интерактивном режиме (online) с CPU даже при использовании программы, независимой от модулей, и даже если в PG не установлено соответствующего проекта.

2.3.5. Информация о CPU (CPU Information)

При интерактивном (online) режиме Вам доступна информация, касающаяся CPU. Команды меню выводятся на экран, если Вы выбираете какой-либо модуль (в интерактивном режиме) или S7-программу (в интерактивном окне проекта). Для получения этой информации служат пункты меню:

- PLC -> Diagnose Hardware (PLC -> Диагностика оборудования),
- PLC -> Module Information (PLC -> Информация о модуле).

Общая информация (такая, например, как версия), диагностический буфер, память (текущее распределение рабочей памяти и загрузочной памяти, сжатие), время цикла (длина последнего цикла, длина самого длинного и длина самого короткого цикла программы), система времени (внутренние часы CPU, синхронизация времени, счетчик рабочего времени), рабочие характеристики (конфигурация памяти, размеры адресных областей, число доступных блоков, SFC и SFB), коммуникации (скорость передачи данных и коммуникационные соединения), стеки в режиме STOP (В-стек, I-стек и L-стек). Для получения этой информации служат пункты меню:

- PLC -> Operating Mode (PLC -> Рабочий режим) - отображение рабочего режима (например, RUN или STOP), изменение рабочего режима;
- PLC -> Clear/Reset (PLC -> Очистка/Сброс) - сброс CPU в STOP-режиме;

- *PLC -> Set Date and Time (PLC -> Установить дату и время)* - установка внутренних часов CPU;
- *PLC -> CPU Messages (PLC -> Сообщения CPU)* - сообщения об асинхронных системных ошибках и сообщения, определяемые пользователем, генерируются в программе с помощью SFC 52 (WR_USMSG), SFC 18 (ALARM_S) и SFC 17 (ALARM_SQ);
- *PLC -> Display Force Values, (PLC -> Отобразить форсированные значения),*
- *PLC -> Monitor/Modify Variables, (PLC -> Мониторинг/модификация переменных).*

2.3.6. Мониторинг и модификация переменных (Monitoring and Modifying Variables)

Есть замечательное средство для отладки пользовательской программы - функция для мониторинга и модификации переменных (Monitoring and Modifying of Variables), использующая VAT-таблицу (таблицу размещения переменных). Состояния сигналов или значения переменных простых типов данных могут быть отображены с помощью этого средства. При наличии доступа к пользовательской программе Вы можете также модифицировать переменные, т.е. изменять состояния сигналов или назначать новые значения.

Для того, чтобы использовать функцию для мониторинга и модификации переменных (Monitoring and Modifying of Variables), Вы должны создать VAT-таблицу (таблицу размещения переменных), содержащую переменные и форматы соответствующих данных. Вы можете генерировать до 255 таблиц переменных (VAT1 - VAT255) и назначить им имена в таблице символов (Symbol Table). Максимальный размер VAT-таблицы составляет 1024 строки с содержанием до 255 символов

Вы можете создать VAT-таблицу автономно (offline), выбрав пользовательскую программу *Blocks* (Блоки), а затем опции меню: *PLC -> Monitor/Modify Variables (PLC -> Мониторинг/модификация переменных)*. Вы можете определять переменные с помощью абсолютной или символьной адресации и выбрать для них тип данных (формат отображения переменной). Для изменения выберите строки, затем: *View -> Display Format (Вид -> Отобразить формат)*, или просто щелкните правой кнопкой мыши на заголовке столбца "Display Format" ("Отобразить формат").

Используйте строки комментариев для разделения таблицы на отдельные секции и придания отдельным частям таблицы заголовков. Вы можете также определять вид таблицы, а именно, какие столбцы должны быть отображены. В любое время Вы можете изменить переменные или формат их отображения, добавить или удалить строки таблицы. Таблица переменных должна быть сохранена в каталоге объекта *Blocks* (Блоки) с помощью опций: *Table -> Save (Таблица -> Сохранить)*.

Для работы с VAT-таблицей, которая была создана автономно (offline), переключите ее в интерактивный режим с помощью опций меню: *PLC -> Connect To (PLC -> Подключить к)*. Вы должны переключать в интерактивный режим каждую таблицу отдельно, а после работы с таблицей - отключать это соединение с помощью опций: *PLC -> Disconnect (PLC -> Разъединить)*.

Для задания условий запуска (**Trigger conditions**) в таблице переменных выберите опции меню: *Variable -> Trigger (Переменная -> Запуск)* - для установки точки запуска (trigger point) и условий запуска (trigger conditions) отдельно для функций мониторинга и модификации. Точка запуска (trigger point) - это такая точка, в которой CPU считывает значения из системной памяти или записывает значения в системную память. Вы должны определить, будет ли считывание или запись происходить один раз или периодически. Если для функции мониторинга и модификации имеются одинаковые условия запуска, то функция мониторинга будет выполняться до функции модификации. Если Вы выбрали для функции модификации точку запуска "Start of cycle" ("Начало цикла"), то переменные будут модифицированы после обновления отображения входов процесса и перед вызовом блока OB 1. Если Вы выбрали для функции мониторинга точку запуска "End of cycle" ("Конец цикла"), то состояния переменных будут выведены после завершения OB1 и перед установкой выходов в соответствии с отображением выходов процесса.

Для выбора функции мониторинга (**Monitoring of Variables**) используйте опции меню: *Variable -> Monitor (Переменная -> Мониторинг)*. Переменные из VAT-таблицы обновляются в соответствии с заданными условиями запуска.

Постоянный мониторинг позволит Вам следить за изменениями значений переменных на экране. Значения отображаются в формате данных, который Вы задаете в столбце Display Format (Формат отображения). Закончить непрерывный процесс мониторинга позволяет кнопка Esc. Использование опций: *Variable -> Update Monitor Values (Переменная -> Обновить отслеживаемые значения)* позволяет однократно обновить наблюдаемые переменные, причем это обновление происходит мгновенно и не зависит от заданных условий запуска.

Для выбора функции модификации переменных (**Modifying of Variables**) для пересылки в CPU измененных значений переменных в соответствии с заданными условиями запуска используйте опции меню: *Variable -> Modify (Переменная -> Модификация)*. Вводите модифицированные значения только в те строки VAT-таблицы, в которых содержатся переменные, которые нужно изменить. Вы можете

распространить комментарий на значение ("закомментировать" значение переменной) с помощью символов "//" или с помощью опций: *Variable -> Modify Value Valid* (*Переменная -> Модификация значения разрешена*); такие значения не берутся в расчет при модификации переменных. Значения отображаются в формате данных, который Вы задаете в столбце Display Format (Формат отображения). Закончить непрерывный процесс модификации можно с помощью кнопки Esc.

Использование опций: *Variable -> Activate Modify Values* (*Переменная -> Активировать модификацию значений*) позволяет однократно модифицировать переменные, причем это происходит мгновенно и не зависит от заданных условий запуска.

2.3.7. Тестирование программы

После выполнения соединения с CPU и загрузки пользовательской программы Вы можете тестировать (отлаживать) программу в целом или по частям, отдельными блоками. Необходимо инициализировать переменные значениями, определенными, например, с помощью модулей симулятора, и оценить информацию отклика, полученного программой в виде значений данных.

Если в результате ошибки CPU переходит в состояние STOP, Вы можете использовать, в частности, информацию о CPU. Большие программы обычно отлаживаются по частям. Если Вам, например, необходимо отлаживать один блок, то загрузите этот блок в CPU и вызовите его в организационном блоке OB1. Если блок OB1 построен таким образом, что программа может быть отлажена фрагмент за фрагментом от начала до конца, то Вы можете выбирать для отладки отдельные блоки или фрагменты программы, используя функции перехода, чтобы миновать разделы, не нуждающиеся в отладке.

С помощью опционного (поставляемого по отдельному заказу) программного обеспечения PLCSIM, Вы можете моделировать CPU в программаторе PG и таким образом отлаживать Вашу программу без дополнительного оборудования.

С помощью функции **"Program Status"** ("Состояние программы") программный редактор обеспечивает дополнительные возможности для тестирования пользовательской программы. С помощью этой функции редактор отображает последовательно для каждой строки программы состояние выбранного Вами регистра. Настройка отображаемых данных выполняется на вкладке "STL" после выбора опций меню: *Options -> Customize* (*Опции -> Установки пользователя*). Столбец "Standard" ("Стандарт") предполагает отображение аккумулятора 1 или значений таймера или счетчика. Для отладки блок программы помещается в пользовательскую память (user memory) CPU, вызывается и обрабатывается в редакторе. Откройте программный блок, например, с помощью двойного щелчка кнопки манипулятора "мышь" на его иконке в интерактивном (online) окне SIMATIC Manager. Редактор запускается и отображает программу, содержащуюся в блоке. Выберите цепочку, которую необходимо отладить. С помощью опций меню: *Debug -> Monitor* (*Отладка -> Мониторинг*) активируйте функцию Program Status. Теперь Вы можете наблюдать состояния адресов, результат логической операции и назначения регистров. С помощью опций меню: *Debug -> Monitor* (*Отладка -> Мониторинг*) Вы можете также деактивировать функцию Program Status.

С помощью опций меню: *Debug -> Call Environment* (*Отладка -> "Обстоятельства вызова"*) Вы можете задать условия запуска функции Program Status. Такие установки Вам потребуются, если блок, который Вы отлаживаете, вызывается более чем один раз в Вашей программе. Вы можете инициировать запись состояний (status recording) или определив порядок вызовов, или сделав его зависимым от открытого блока данных. Если блок вызывается только один раз, то выберите "No Condition" ("Нет условий").

Вы можете модифицировать переменные, используя функцию Program Status. Выберите адрес данных, которые должны быть модифицированы, затем выберите опции: *Debug -> Modify Address* (*Отладка -> Модифицировать адрес*). Запись информации функции Program Status требует дополнительного времени в цикле выполнения программы. Поэтому Вы можете выбирать один из двух рабочих режимов для отладки программы: "test mode" (режим отладки) и "process mode" (режим обработки процесса). Первый из указанных режимов (режим отладки) позволяет использовать все функции отладки без ограничения. Этот режим выбирается, например, для отладки блоков без подключения к системе, так как отдельные функции отладки сильно увеличивают время выполнения цикла сканирования программы.

В режиме "process mode" необходимо стремиться минимизировать удлинение цикла сканирования из-за функций отладки, поэтому здесь накладываются определенные ограничения, например, на программные циклы (не все проходы циклов программы отображаются). Отдельные CPU позволяют установку рабочего режима на вкладке "Protection" ("Защитный режим") при параметризации CPU. Если при параметризации CPU был установлен режим отладки, то Вы сможете изменить режим работы только во время повторной процедуры параметризации. Другими словами, режим может быть изменен в диалоговом окне.

Установка рабочего режима отображается с помощью опций: *Debug -> Operation* (*Отладка -> Работа*).

Для блоков, написанных на языке STL, некоторые CPU позволяют выполнять отладку программы оператор за оператором в пошаговом режиме **"Single-step mode"**. CPU находится в режиме HOLD; в целях безопасности периферийные выходы заблокированы. Используя точки прерывания (breakpoints), Вы можете остановить программу в любой момент и отлаживать в пошаговом режиме (step-by-step).

Должен быть установлен режим отладки, а программа запущена в режиме online.

Для установки точки прерывания (breakpoint) установите курсор в строке с нужным оператором и выберите опции: *Debug -> Set Breakpoint (Отладка -> Установка точки прерывания)*. Для отладки выберите опции меню: *Debug -> Breakpoints Active (Отладка -> Активировать точки прерывания)*; эта команда вызовет перенос точек прерывания в CPU и активирует их. CPU необходимо перезапустить, в процессе обработки программы CPU при достижении точки прерывания переходит в состояние паузы HOLD. При этом в специальном окне будет отображено текущее содержание регистров. Теперь Вы можете запустить программу для выполнения в построчном (в пошаговом) режиме, выбрав опции меню: *Debug -> Execute Next Statement (Отладка -> Выполнить следующий оператор)*. При этом выполнение программы будет прерываться на каждом операторе, и при этом будет отображаться текущее содержание регистров. Если в текущем операторе производится вызов блока, Вы можете продолжить пошаговую обработку программы в этом блоке, если выберете опции: *Debug -> Execute Call (Отладка -> Выполнять вызов)*.

С помощью опций меню: *Debug -> Resume (Отладка -> Продолжить)* программа выполняется с нормальной скоростью, пока не встретит следующую точку прерывания. Блоки, содержащие точки прерывания, не могут быть изменены или перезагружены в интерактивном режиме (online). Все точки прерывания сначала должны быть удалены. Все точки прерывания также должны быть удалены при выходе из режима отладки с точками прерывания.

После удаления всех точек прерывания с помощью опций меню *Debug -> Resume (Отладка -> Продолжить)* CPU снова переключается в режим RUN.

3. SIMATIC S7-программа

В целом программное обеспечение для CPU состоит из операционной системы (operating system) и пользовательской программы (user program).

Операционная система - это совокупность всех инструкций и деклараций, которые управляют системными ресурсами и процессами, использующими эти ресурсы. Операционная система включает в себя такие функции как резервирование данных в случае сбоя электропитания, активация приоритетных классов и т.п. Операционная система - это такой компонент CPU, к которому Вы, как пользователь, не имеете доступа в режиме записи. Тем не менее, Вы можете перезагружать операционную систему с модуля памяти, например, в случае обновления программы.

Пользовательская программа (user program) - это совокупность всех инструкций и деклараций (в данном случае - программных элементов), для обработки сигналов, с помощью которых установка (процесс) регулируется в соответствии с определенной задачей управления.

Пользовательская программа может состоять из программных разделов, которые обрабатываются в CPU в зависимости от конкретного события. Одним из таких событий может быть, например, запуск системы автоматического управления, прерывание или обнаружение программной ошибки (см. рис. 3.1). Программы, назначенные для обработки событий, разделяются по приоритетным классам (*priority class*), с помощью которых определяется порядок обработки отдельных разделов ("mutual interruptibility" - система взаимных прерываний) программы в случаях, когда происходит одновременно несколько событий. Классы приоритетов даны в таблице 3.1.

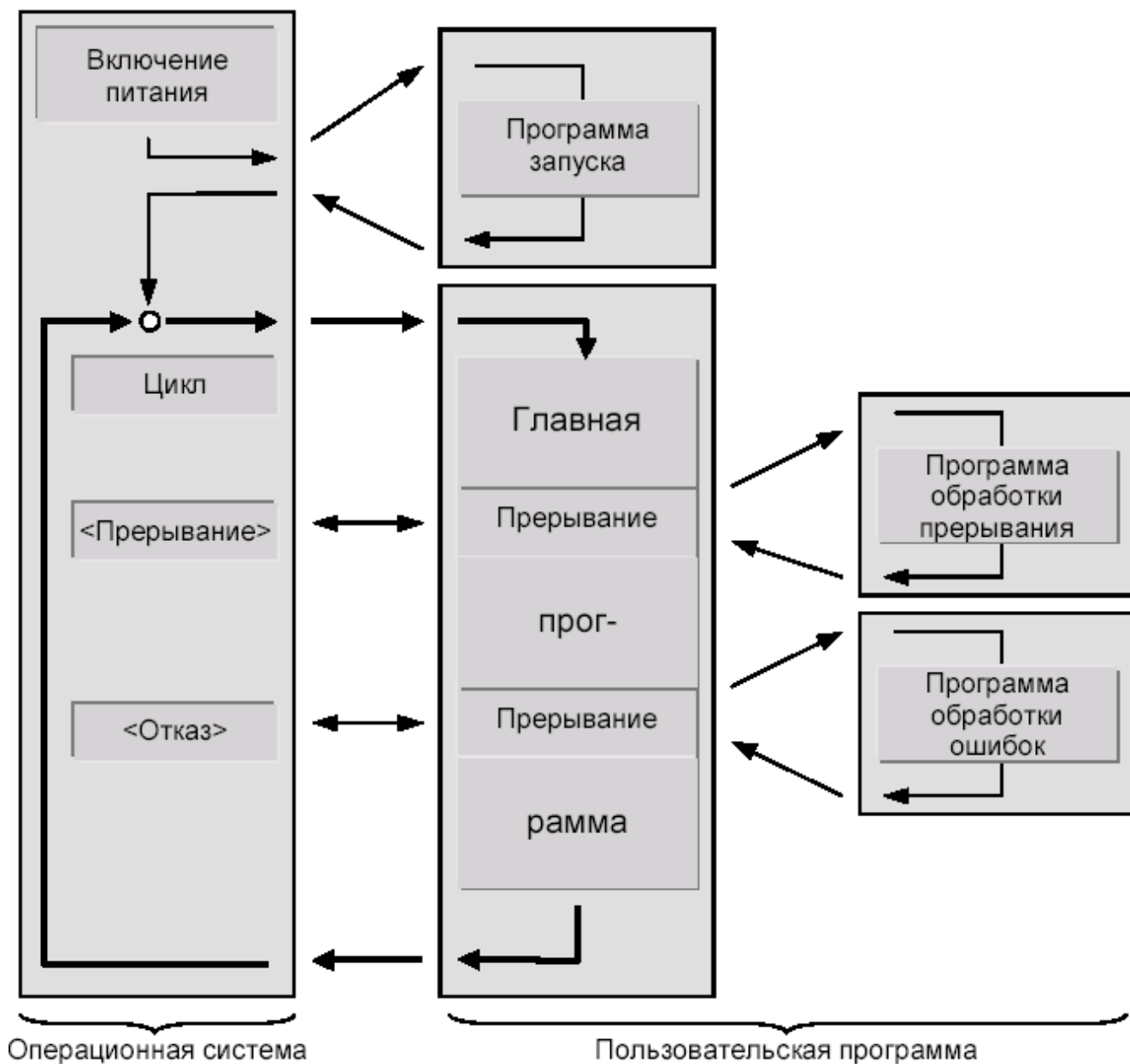


Рис. 3.1. Обработка пользовательской программы

В некоторых классах приоритетов Вы можете изменять заданный приоритет при параметризации CPU. В таблице 3.1 для организационных блоков показаны возможные нижний и верхний приоритетные классы. Каждый CPU имеет свой диапазон значений для нижнего и верхнего приоритетных классов. В данном обзоре представлены отдельные типы CPU.

Организационный блок OB 90 (фоновая обработка только для S7-400) может выполняться вместо OB 1 и, как в случае с OB 1, его обработка может быть прервана любыми прерываниями и ошибками. Программа запуска может быть в организационном блоке OB 100 (полный перезапуск) или в организационном блоке OB 101 (теплый перезапуск); она имеет приоритетный класс 27. Асинхронные ошибки, происходящие в программе запуска, имеют приоритетный класс 28. Диагностические прерывания рассматриваются как асинхронные ошибки.

Вы должны определить, какие доступные приоритетные классы Вы будете использовать при параметризации CPU. Неиспользуемые приоритетные классы (организационные блоки) должны получить приоритет 0. Соответствующие организационные блоки должны быть запрограммированы для всех используемых приоритетных классов; иначе CPU вызовет OB 85 ("Program Processing Error" - "Ошибка выполнения программы") или перейдет в режим STOP.

Любой организационный блок запускается только операционной системой контроллера. Блок, в свою очередь может вызывать функции (FC) и функциональные блоки (FB)

Программа с наинизшим приоритетом (OB1) является главной программой, которая циклически обрабатывается CPU. События могут прерывать главную программу в любой момент, после чего CPU выполнит связанную с прерыванием обслуживающую программу или программу обработки ошибок и затем вновь вернет управление в главную программу. Специальный организационный блок

Таблица 3.1. Организационные блоки SIMATIC S7

Организационный блок	Условия вызова	Приоритет	
		по умолчанию	возможные изменения
OB 1	Периодически вызывается операционной системой	1	Нет
OB 10 - OB 17	В определенное время суток или через равные промежутки времени (например, ежемесячно)	2	2 - 24
OB 20 - OB 23	По истечении запрограммированного времени; управление из пользовательской программы	3 - 6	2 - 24
OB 30 - OB 38	Регулярный вызов через запрограммированные интервалы времени (например, каждые 100 мс)	7 - 15	2 - 24
OB 40 - OB 47	Вызов по сигналу прерывания от I/O модулей	16 - 23	2 - 24
OB 60	Вызов по приходу события в мультипроцессорном режиме; управление из пользовательской программы	25	Нет
OB 70, OB 72, OB 73	В случае потери резервирования из-за ошибок I/O	25	2 - 26
	В случае ошибки резервирования CPU	28	2 - 28
	В случае ошибки резервирования коммуникаций	25	24 - 26
OB 80	В случае ошибок, не связанных с выполнением программы (например, time error, диагностическое прерывание, прерывание удаления модуля, отказ стойки / станции)	26 ²⁾	26
OB 81-OB 84, 86, 87		26 ²⁾	2 - 26
OB 85		26 ²⁾	24 - 26
OB 90	Продолжительность минимального цикла еще не достигнута	29 ¹⁾	Нет
OB 100, 101, 102	Программа запуска	27	Нет
OB 121, OB 122	В случае ошибок, связанных с выполнением программы	Приоритет OB, вызвавшего ошибку	

1) см. текст

2) при запуске - 28

(OB) соответствует каждому событию. Организационные блоки имеют приоритетные классы в пользовательской программе. Если происходит событие, CPU активизирует соответствующий ему организационный блок. Организационный блок (OB) - это часть пользовательской программы, которую Вы сами можете написать. Общая структура программы показана на рис. 3.2.

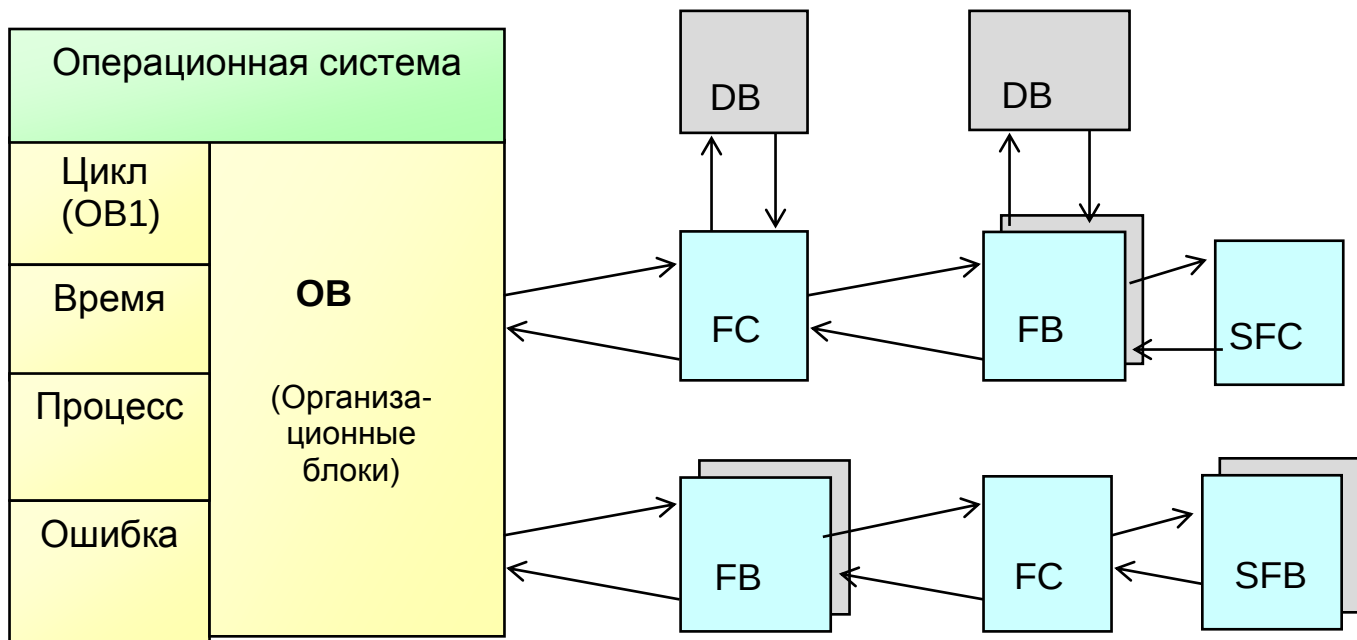


Рис. 3.2. Общая структура выполнения программы

Перед тем, как CPU начнет обработку главной программы, он выполняет программу запуска (startup routine). Эта программа может быть запущена такими событиями как включение питания, поворот переключателя режимов на передней панели CPU или с помощью программатора PG.

Обработка программы, следующая за выполнением программы запуска, в системе S7-300 всегда начинается с начала главной программы ("complete restart" - "полный перезапуск"), а в системе S7-400

может также использоваться режим, когда продолжается сканирование программы с точки, в которой оно было прервано ("warm restart" - "теплый перезапуск").

Главная программа располагается в организационном блоке OB 1, который используется практически во всех пользовательских программах. После завершения выполнения OB 1, что соответствует окончанию выполнения программы, CPU возвращает управление в операционную систему и после вызова различных функций операционной системы, таких как обновление отображений процесса, вновь вызывает для выполнения OB 1 (см. Рис. 3.3).

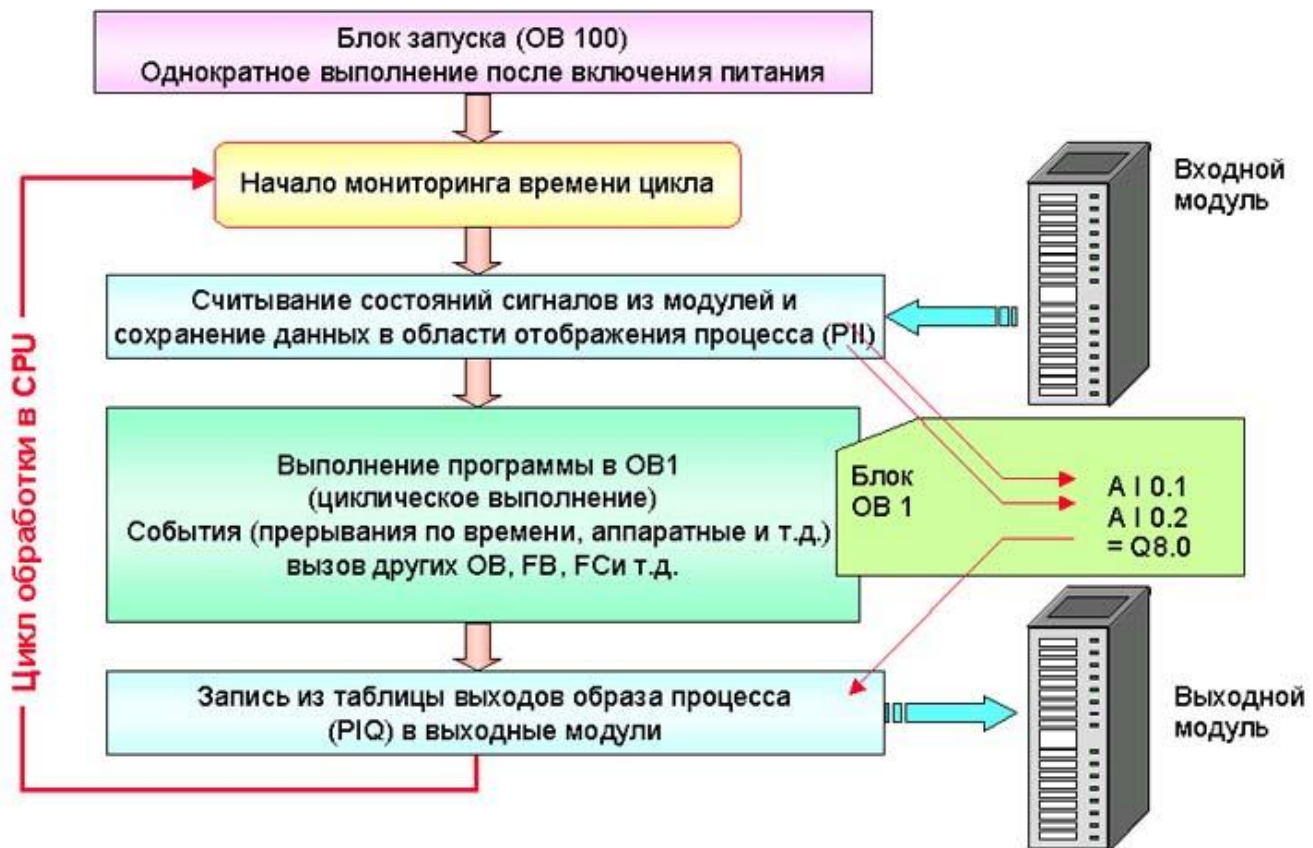


Рис. 3.3.

События, которые могут вмешиваться в работу программы, - это прерывания и ошибки. Источником прерываний могут стать либо процесс (аппаратное прерывание), либо CPU: прерывания по времени ["watchdog", то есть таймерные], прерывания по времени суток ["time-of-day"] и т.д. Что касается ошибок, то они подразделяются на синхронные и асинхронные. Асинхронные ошибки - это такие ошибки, которые не зависят от выполняемой программы, например, это может быть сбой питания на устройстве расширения или такое событие, как удаление модуля из стойки. Синхронные ошибки - это такие ошибки, которые возникают при выполнении программы, например, ошибка будет зафиксирована при попытке доступа к несуществующему адресу или ошибка может произойти при преобразовании типов данных. Типы и номера регистрируемых событий и соответствующих организационных блоков определяются типом CPU; не каждый CPU способен обрабатывать все события, возможные в STEP 7.

Операционная система CPU обычно использует параметры, принятые по умолчанию. Вы можете изменить эти установки при параметризации CPU с помощью утилиты конфигурирования оборудования HW Config для того, чтобы параметры системы удовлетворяли Вашим особым требованиям. Вы можете изменить эти параметры в любое время.

Каждый CPU имеет свой собственный особый набор параметров. В приведенном ниже списке представлен обзор всех параметров STEP 7 и их наиболее важные установки.

- Startup (параметры запуска) – характеристика типа запуска ("cold restart" ["холодный перезапуск"] / "warm restart" ["теплый перезапуск"]); мониторинг сигналов "Ready" или параметризации модуля; максимальная продолжительность времени, которое может произойти до момента "теплого перезапуска".
- Cycle/clock memory (цикл/такты меркеры) - включение/выключение циклического обновления отображения процесса; задание продолжительности периода мониторинга и минимальной продолжительности времени цикла; продолжительность времени цикла в процентах для коммуникаций; число тактовых меркеров; размер области отображения процесса.

- Retentive memory (перманентная память) - число перманентных меркеров, таймеров и счетчиков; определение перманентных областей для блоков данных.
- Memory (память) - максимальное количество временных локальных данных в приоритетных классах (в организационных блоках); максимальный размер L-стека и число коммуникационных заданий.
- Interrupts (прерывания) - спецификация приоритета аппаратных прерываний, прерываний с задержкой (time-delay interrupts), асинхронных ошибок и коммуникационных прерываний.
- Time-of-day Interrupts (прерывания по времени суток) - спецификация приоритета, спецификация стартового времени и периодичности.
- Cyclic Interrupts (циклические прерывания) - спецификация приоритета, спецификация времени цикла и фазового смещения.
- Diagnostics/Clock (диагностические прерывания/системные часы) - индикация причины перехода в состояние STOP; тип и интервал синхронизации времени; коэффициент коррекции.
- Protection (параметры доступа к программам) - спецификация уровня защиты, задание пароля.
- Multicomputing (параметры мультипроцессорного режима) - определение числа CPU.
- Integrated I/O (параметры встроенных I/O) - активация и параметризация встроенных I/O. При запуске CPU загружает заданные пользователем параметры вместо параметров, принятых по умолчанию. Параметры, определенные пользователем остаются в силе до тех пор, пока они не будут заменены.

3.1. Блоки

Чтобы сделать Вашу программу более легкой для чтения и понимания, Вы можете разбить ее на такое количество частей, какое Вы пожелаете. Языки программирования STEP 7 поддерживают такой подход к программированию, обеспечивая Вас необходимыми функциями. Каждая такая часть программы должна быть самодостаточной (self-contained) и должна решать технологическую или функциональную задачу.

Рассматриваемые части программы называются "блоками". Блок - это часть программы пользователя, характеризующаяся своими собственными задачами, структурой или функциональным назначением.

Язык программирования STL использует различные типы блоков для разных задач:

- User blocks (пользовательские блоки) - это блоки, содержащие пользовательскую программу и пользовательские данные;
- System blocks (системные блоки) - это блоки, содержащие системную программу и системные данные;
- Standard blocks (стандартные блоки) - это готовые к использованию блоки, такие, например, как драйверы для функциональных модулей (FM) или коммуникационных процессоров (CP).

Для больших и сложных программ "структурирование" - подразделение программы на **пользовательские блоки** рекомендуется и отчасти является необходимостью. Вы можете выбирать среди различных типов блоков те или иные, в зависимости от условий применения.

Организационные блоки OB (Organization blocks) служат своеобразным интерфейсом между операционной системой и пользовательской программой. Операционная система CPU вызывает организационные блоки при возникновении событий, например, аппаратного прерывания или прерывания времени суток.

Главная программа находится в организационном блоке OB 1. Остальные организационные блоки имеют постоянные назначенные номера и предназначены для обработки событий, для которых они вызываются.

Функциональные блоки FB (Function blocks) являются частями программы, вызов которых может быть запрограммирован с помощью параметров блока. Они обладают областью памяти для переменных (variable memory), которая расположена в блоке данных. Этот блок данных постоянно назначен функциональному блоку, или, точнее, *вызову* функционального блока. Возможно даже назначение нескольких блоков данных (с одинаковой структурой данных, но содержащих разные значения) функциональному блоку. Такой постоянно назначенный блок данных называется *экземплярным блоком данных* (instance data block), а совокупность вызова функционального блока и экземплярного блока данных называется *экземпляром вызова* (call instance) или, для краткости, "экземпляром" ("instance"). Функциональные блоки могут также хранить свои переменные в экземплярном блоке данных вызывающего функционального блока; тогда такой экземплярный блок данных называется "локальным экземпляром" ("local instance").

Функции FC (Functions) используются для программирования часто повторяющихся или сложных функций автоматизации (automation functions). Функциям могут назначаться параметры. Функции могут возвращать значение (значение вызванной функции) в вызывающий блок. Причем значение функции

- необязательный параметр. Кроме функционального значения функция может иметь другие выходные параметры. Функции не имеют назначенных блоков данных.

Блоки данных DB (Data blocks) содержат данные Вашей программы. Описывая блоки данных, Вы определяете - в какой форме данные будут сохраняться (в котором блоке, в каком порядке и с каким типом данных). Существует два способа использования блоков данных: как блоки глобальных данных (global data blocks) и как экземплярные блоки данных (instance data blocks). Блоки глобальных данных в пользовательской программе являются, как говорится, "свободными" ("free") блоками данных и не назначаются кодовому блоку. Экземплярные блоки данных, однако, назначаются функциональному блоку и сохраняют часть локальных данных этого функционального блока. Максимальное число для каждого типа блоков и размер этих блоков определяются типом CPU.

Число организационных блоков и их номера фиксированы; они назначаются операционной системой CPU. Блокам других типов Вы можете самостоятельно назначать номера внутри определенных пределов. Также Вы можете выбрать для каждого блока имя (символ) в таблице символов, с тем, чтобы ссылаться на блок по символьному имени.

Системные блоки (System blocks) являются компонентами операционной системы. Они могут содержать программы (системные функции SFC или системные функциональные блоки SFB) или данные (системные блоки данных SDB). Системные блоки предоставляют множество важных системных функций, доступных пользователю, таких, например, как функции управления внутренними часами CPU или различные коммуникационные функции. Вы можете вызывать SFC и SFB, но Вы не можете ни изменить эти блоки, ни запрограммировать их самостоятельно. Системные блоки не занимают места в пользовательской памяти (user memory); только вызовы блоков и экземплярные блоки данных для SFB располагаются в пользовательской памяти. Системные блоки данных (SDB) содержат информацию о таких вещах, как конфигурация автоматизированной системы или параметры модулей. Система STEP 7 самостоятельно генерирует эти блоки и управляет ими. Тем не менее, Вы можете определять их содержимое, например, когда Вы конфигурируете станции. Как правило, системные блоки данных размещаются в загрузочной (load) памяти. Пользователь не имеет доступа к ним из своей программы.

В дополнение к функциям и функциональным блокам, которые Вы можете создавать самостоятельно, Вы можете использовать готовые для применения блоки, так называемые **"стандартные блоки"** ("Standard blocks"). Эти блоки могут быть получены или на различных носителях, или могут быть в составе библиотек из комплекта поставки ПО STEP 7 (например, IEC-функции или функции для S5/S7 конвертирования).

3.1.1. Свойства блоков

Свойства блоков (Block Properties) или атрибуты содержатся в заголовке блока. Вы можете увидеть и изменить атрибуты блока в редакторе с помощью опций меню: *File -> Properties (Файл -> Свойства)*.

На вкладке "General - Part 2" ("Общие - часть 2") показано распределение памяти для блока в байтах:

- Local Data (временные локальные данные - размещение стека локальных данных),
- MC 7 - размер блока (только код),
- Load memory requirement (требования к загрузочной памяти),
- Work memory requirement (требования к рабочей памяти).

Атрибут *"Know-how protection"* ("защита технологии") используется для защиты блока. Если блок защищен с помощью установки этого атрибута, то программа из этого блока не может быть отображена, распечатана или изменена. В редакторе можно будет увидеть только заголовок блока и таблицу объявления переменных (declaration table) с параметрами блока. При вводе текста в исходный файл Вы можете защитить каждый блок сами с помощью ключевого слова `KNOW_HOW_PROTECT`. Если Вы защитили блок, то никто (даже Вы) не сможет увидеть компилированный вариант этого блока (Вы должны обеспечить при этом безопасное место хранения для исходного файла программы!).

Заголовок блока любого стандартного блока (standard block), который предоставляется фирмой Siemens, содержит атрибут *"Standard Block"*. Атрибут *"DB is write-protected in the PLC"* ("DB в PLC доступен только для чтения") используется только для блоков данных. Установка этого атрибута приведет к тому, что Вы сможете только считывать данные блока в Вашей программе. При попытке записи в защищенный блок данных выводится сообщение об ошибке. Такой вариант защиты (защита от записи) нельзя путать с защитой блока. Блок данных с защитой блока может быть считан и перезаписан в пользовательской программе, но его данные нельзя отобразить ни с помощью программатора PG, ни с помощью устройств наблюдения оператора.

Адресам данных в блоке данных могут быть назначены начальные значения (initial value) и фактические значения (actual value). Если блок данных загружен в CPU, то начальные (initial) значения пересылаются в загрузочную (load) память, а фактические значения (actual) пересылаются в рабочую (work) память.

Каждое изменение значения в соответствии с адресацией данных в программе предопределяет изменение соответствующего фактического значения.

Если Вы выгружаете блок данных из CPU, значения данных блока берутся из рабочей (work) памяти, в которой хранятся все фактические (actual) значения данных. Вы можете наблюдать фактические (actual) значения данных во время их считывания с помощью опций меню: *View -> Data View (Bud -> Bud данных)*. Если Вы модифицируете фактические (actual) значения данных в блоке данных, а затем вновь записываете блок в CPU, то модифицированные значения поступают в рабочую (work) память.

Если в качестве загрузочной памяти используется модуль памяти типа флэш EPROM, то блоки из модуля памяти переносятся в рабочую (work) память после сброса памяти CPU. При этом блок памяти сохраняет первоначально запрограммированные в нем значения данных. То же самое происходит при включении питания при отключенной резервной батарее. Для S7-300 Вы можете предотвратить загрузку первоначальных значений данных, если Вы объявите эти области данных ретентивными (retentive).

Блок данных, созданный с активированным свойством "UNLINKED" ("несвязанный"), не переносится в рабочую память; он остается в загрузочной памяти. Блок данных с активированным свойством "UNLINKED" может быть считан только с помощью функции SFC 20 (BLKMOV). Вы не сможете записывать данные в блоки, находящиеся в загрузочной памяти.

Прочие спецификации вкладки "General - Part 2" ("Общие - часть 2") окна свойств блока:

- *Name (Имя)* идентифицирует блок; это не то же самое, что и символьный адрес: разные блоки могут иметь одинаковое имя;
- *Family (Семья - имя группы)* позволит Вам назначить общие характеристики для группы блоков;
- *Author (Автор)* идентифицирует создателя блока.

Идентификаторы блока *Name (Имя)* и *Family (Семья - имя группы)* отображаются при вставке блоков и при выборе блоков в диалоговом окне каталога элементов программы.

Атрибуты *Name (Имя)*, *Family (Семья - имя группы)*, *Author (Автор)* могут содержать до 8 символов (здесь могут применяться следующие символы: буквы, цифры и знак подчеркивания).

Атрибут *Version (Версия)* вводится дважды двумя цифрами: от 0 до 15 на вкладке "General - Part 1" ("Общие - часть 1").

Редактор записывает дату изменения блока в две отметки времени: для кодового блока и для интерфейса, т.е. для блока параметров и для статических локальных данных.

Примечание: надо отметить, что дата изменения интерфейса должна быть такой же или более ранней, нежели дата изменения программного кода вызывающего блока. Если это условие не выполняется, то при выводе на экран вызывающего блока редактор сигнализирует об ошибке "time stamp conflict" ("конфликт отметок времени").

В окне свойств объекта *Blocks* (режим "offline") на вкладке "Blocks" ("Блоки") Вы можете назначить приоритет для одного из типов адресации (абсолютной или символьной) для уже сохраненных блоков, если эти блоки отображены. Перед этим все изменения в таблице символов или все назначения блоков глобальных данных должны быть сохранены. По умолчанию принята установка: "Absolute address has priority" (абсолютная адресация имеет приоритет. Такая установка означает, что если делаются изменения в таблице символов, то остается в силе абсолютная адресация, а символьные адреса соответственно изменяются.

Если задана установка: "Symbol has priority" (символьная адресация имеет приоритет), то это означает, что остается в силе символьная адресация, а абсолютные адреса изменяются в программе.

Пример. Пусть таблица символов содержит следующую информацию:

I 1.0 "Limit_switch_up" ("Верхний концевой переключатель")
I 1.1 "Limit_switch_down" ("Нижний концевой переключатель"),
и в программе скомпилированного блока сканируется вход I 1.0:

A I 1.0 "Limit_switch_up"

Если задана установка: "Symbol has priority", а в таблице символов поменять значения выше указанных входов следующим образом:

I 1.0 "Limit_switch_down" ("Нижний концевой переключатель");
I 1.1 "Limit_switch_up" ("Верхний концевой переключатель"),
то программа будет содержать строку:

A I 1.1 "Limit_switch_up",

а если задана установка: "Absolute address has priority", то программа будет содержать строку:

A I 1.0 "Limit_switch_down".

Если задана установка: "Symbol has priority", то инкрементно программируемые блоки с символической адресацией сохраняют свои символы в случае изменений в таблице символов. В этом случае такой блок с уже готовой программой может быть перекомпонован (rewired) с помощью изменения назначения адресов.

Примечание: такая перекомпоновка не может выполняться автоматически, так как скомпилированные блоки содержат исполняемый MC7-код с абсолютной адресацией. Изменения делаются только в соответствующих блоках при поступлении соответствующего сообщения, после которого они могут быть открыты и окончательно вновь сохранены.

Блоки могут быть созданы или скомпилированы в виде версии 1 или в виде версии 2. Это имеет практическое значение только для функциональных блоков. Если активировано свойство "multi-instance capability" ("несколько экземпляров DB для функционального блока"), что, кстати, является обычным случаем, то мы имеем дело с блоком версии 2. Если свойство "multi-instance capability" выключено, то Вы не сможете вызвать этот функциональный блок как локальный экземпляр, также как Вы не сможете вызвать другой функциональный блок из этого блока как локальный экземпляр.

Функциональный блок версии 1 имеет преимущество, заключающееся в ограничении использования экземпляра блока данных в случае косвенной адресации (имеет значение только при STL-программировании).

На вкладке "Calls" ("Вызовы") Вы увидите список всех блоков, вызываемых в данном блоке с отметками времени для кодовых блоков и интерфейса.

На вкладке "Attributes" ("Атрибуты") показаны системные атрибуты блока. С помощью системных атрибутов осуществляется координация и управление функциями разных приложений, например, в системе управления SIMATIC PCS7.

Длина пользовательской программы содержится в свойствах (Properties) в автономном каталоге *Blocks* (Блоки). Для доступа к этим данным выберите *Blocks* (Блоки) и используйте опции: *Edit -> Object Properties* (Правка -> Свойства объекта). Теперь на вкладке "Blocks" ("Блоки") Вам доступна информация "Size in work memory" (Размер в рабочей памяти) и "Size in load memory" (Размер в загрузочной памяти).

Примечание: примите во внимание, что конфигурационные данные (системные блоки данных) не учитываются при указании размера программы в загрузочной (load) памяти. Открыв каталог *Blocks* (Блоки), Вы можете увидеть требования к загрузочной (load) памяти для системных данных в деталях (представленных в табличной форме). В строке состояния утилиты SIMATIC Manager указывает суммарный объем памяти для всех блоков, которые Вы выберете с нажатой клавишей Ctrl.

С помощью программатора PG, подключенного интерактивно (online), при использовании утилиты SIMATIC Manager Вы можете найти текущие назначения памяти CPU на вкладке "Memory" ("Память"), используя опции меню: *PLC -> Module Information* (PLC -> Информация о модуле).

Редактор программ генерирует контрольную сумму (Checksum) для всех блоков пользовательской программы и сохраняет ее в свойствах объекта. Идентичные программы имеют одинаковую контрольную сумму, контрольная сумма изменяется при любом изменении программы. Контрольная сумма также генерируется для системных данных. Для доступа к контрольным суммам при помощи утилиты SIMATIC Manager выберите каталог *Blocks* (Блоки) и используйте меню: *Edit -> Object Properties* (Правка -> Свойства объекта).

3.1.2. Интерфейс блоков (Block Interface)

Таблица объявления переменных содержит интерфейс блока с остальной программой. Он состоит из параметров блока (входные, выходные и входные / выходные параметры), а также статических локальных данных (для функциональных блоков). Временные локальные данные не принадлежат интерфейсу блока. Интерфейс блока определяется в таблице объявления переменных, и эти переменные инициализируются при вызове блока (см. раздел "Параметры блоков").

Редактор программ проверяет, соответствует ли инициализация параметров вызываемого блока интерфейсу вызываемого блока. Для этого редактор использует метки времени: интерфейс вызываемого блока должен иметь более раннюю временную метку, чем код вызывающего блока, что означает, что последние изменения интерфейса должны быть выполнены раньше его объединения с блоком. Редактор программ обновляет метку времени интерфейса при изменении числа параметров, или при изменении типа данных, или при изменении значений параметров, принимаемых по умолчанию.

Если интерфейс вызываемого блока имеет более позднюю временную метку, чем код вызывающего блока, возникает "конфликт временных меток" ("Time stamp conflict"). Так, Вы получите "конфликт временных меток", если вновь откроете уже скомпилированный блок. В этом случае редактор выделит некорректный вызов блока красным цветом. Конфликт временных меток также возникнет, если Вы, например, измените интерфейсы блоков, которые уже вызывались в других блоках, или если Вы объедините блоки из разных программ в новую программу, или если Вы перекомпилируете раздел полной программы из исходного файла.

Тем не менее, конфликт интерфейса, в общем описываемый как "конфликт временных меток", может также иметь другие причины. Он может случиться, если вызванный или адресованный (referenced)

блок имеет более позднюю временную метку (younger), чем вызывающий блок. Ниже представлены примеры возможных случаев "конфликта временных меток"):

- Интерфейс вызываемого блока имеет более позднюю временную метку (younger), чем код вызываемого блока.
- интерфейс инициализации не согласован с интерфейсом блока;
- функциональный блок имеет более позднюю временную метку (younger), чем его экземплярный блок данных (экземпляр DB генерируется на основе описания интерфейса функционального блока и должен, следовательно, иметь более позднюю временную метку, чем метка функционального блока, или их метки должны быть синхронны);
- интерфейс локального экземпляра имеет более позднюю временную метку, чем вызывающий экземпляр (касается функциональных блоков);
- пользовательский тип данных UDT имеет более позднюю временную метку, чем блок, переменные которого объявлены как UDT; это может быть любой блок, включая блок данных или другой UDT.

Редактор программ обеспечивает возможность исправления некорректных вызовов блока или UDT-приложений при выборе команд меню: *Edit -> Block Call -> Update (Правка -> Вызов блока -> Обновить)*. Для случая одинаковых имен, типов данных или местоположения редактор может найти правильные назначения в большинстве случаев. Если этого не произошло, то Вы должны выполнить корректировку вручную. В любом случае Вы должны проверить правильность выполненной корректировки.

Редактор программ лишь информирует о наличии "конфликта временных меток", если Вы открываете блок, содержащий этот самый "конфликт временных меток". Если необходимо проверить программу целиком, Вы можете использовать функцию проверки консистентности блока "Check Block Consistency". Эта функция снимает большинство конфликтов интерфейса и указывает на места в программе, требующие редактирования. Для выполнения проверки на консистентность данных выберите каталог *Blocks (Блоки)* и затем опции меню: *Edit -> Check Block Consistency (Правка -> Проверки консистентности блока)*. Редактор программ генерирует данные, требующиеся для такой проверки, начиная с системы STEP 7 V5.0 SP3.

Пользовательская программа может быть скомпилирована в системе STEP 7 более ранней версии или содержать блоки, скомпилированные в системе STEP 7 более ранней версии. Вам необходимо будет проверить это, если соответствующая информация не отображается в окне функции "Check Block Consistency", выбрав в этом окне: *Program -> Compile (Программа -> Компилирование)*.

Редактор программ отобразит процесс выполнения задачи и результат проверки на консистентность в окне результата ("1:Compile"). Такая проверка консистентности не может быть использована для программ, находящихся в библиотеках.

Отношения вызванных или адресованных блоков отображаются в форме древовидной диаграммы. Вы можете выбирать между двумя представлениями, указанными ниже.

"Дерево ссылок" (*reference tree*) представляет связи аналогично отображению структуры программы: слева расположены вызывающие блоки, а правее расположены вызванные ими блоки.

"Дерево подчиненности" (*dependency tree*) представляет связи, начиная от всех вызванных или адресованных блоков. Эти блоки расположены в левом ряду, а правее расположены вызывающие их блоки.

Для обоих представлений случай появления знака восклицания (!) будет означать необходимость исправления и компиляции соответствующего блока.

Для обеспечения безостановочной работы контроллера необходимо либо загружать связанные блоки в режиме Stop, либо загружать в контроллер сначала используемые или вызываемые блоки, а только после этого – вызывающие.

3.2. Адресация переменных (Addressing Variables)

При адресации переменных Вы можете выбирать способ адресации из двух основных вариантов: абсолютная адресация (absolute addressing) или символьная адресация (symbol addressing). При абсолютной адресации используются численные адреса, начиная с нулевого (0) адреса для каждой адресной области. При символьной адресации используются символьные (состоящие из букв и цифр) имена, которые Вы сами задаете в таблице символов (Symbol Table) для глобальных адресов или в разделе объявления переменных (declaration section) для внутриблочной адресации. Расширением абсолютной адресации является косвенная адресация (indirect addressing), при которой адреса (местоположение) в памяти высчитываются во время выполнения программы.

3.2.1. Абсолютная адресация переменных

Переменные простых типов (т.е. состоящие из 1, 8, 16, 32 битов), могут быть адресованы с использованием абсолютной адресации (absolute addressing).

Абсолютные адреса входов и выходов рассчитываются, исходя из начального адреса модуля, который Вы установили или должны установить в таблице конфигурации (configuration table), и типа сигнала, подключаемого к модулю. Подключаться могут как дискретные, так и аналоговые сигналы.

Дискретный сигнал содержит один бит информации. Цифры 1 и 0 называются двоичными цифрами (binary digits) или просто битами. Примерами дискретных сигналов являются входные сигналы от конечных выключателей, кнопок и т.п., которые поступают на дискретные входные модули, и выходные сигналы, управляющие лампами, контакторами и т.п., которые поступают на дискретные выходные модули.

Аналоговый сигнал содержит 16 бит информации, и соответствует "каналу" ("channel"), который занимает в контроллере машинное слово (word), т.е. 2 байта (см. ниже). Аналоговые входные сигналы (например, напряжения от терморезисторов) поступают в аналоговые входные модули, оцифровываются и после этого становятся доступными для обработки в контроллере в виде 16-разрядного сигнала (16 информационных битов). С другой стороны, 16-разрядный сигнал может управлять аналоговым индикатором посредством преобразования в аналоговом выходном модуле в аналоговый сигнал (например, ток).

Динамическому диапазону изменения сигнала соответствует динамический диапазон изменения переменной, в виде которой сигнал сохраняется и обрабатывается. Динамический диапазон изменения сигнала и интерпретация этого сигнала (например, относительное положение), взятые вместе, определяют тип данных (*data type*) для соответствующей переменной.

Дискретные сигналы сохраняются в переменных типа BOOL (булева переменная), аналоговые сигналы - в переменных типа INT (целая переменная).

Определяющим фактором для адресации переменной является ее тип, от которого зависит требуемая величина области памяти для размещения переменной. В системе STEP 7 существуют 4 типа данных для абсолютной адресации (рис. 3.4):

- 1 бит - тип данных BOOL,
- 8 битов - тип данных BYTE или другой 8-битовый тип данных,
- 16 битов - тип данных WORD или другой 16-битовый тип данных,
 - 32 бита - тип данных DWORD или другой 32-битовый тип данных.

На переменные типа BOOL ссылка производится посредством идентификатора адреса, номера байта и отделенного десятичной точкой номера бита. Нумерация байтов начинается с нуля (0) в каждой адресной области. Верхнее предельное значение номера байта определяется типом CPU. Биты внутри байтов нумеруются от 0 до 7 справа на лево.

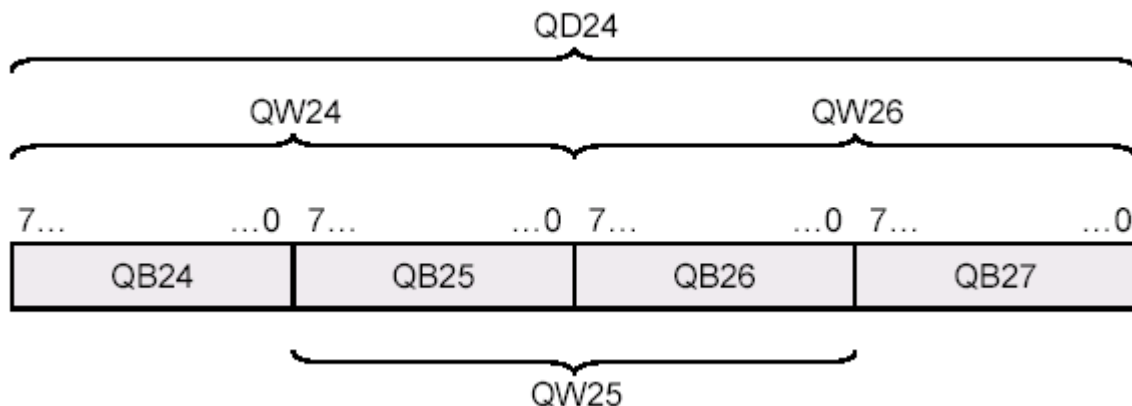


Рис. 3.4. Биты и байты в машинных словах и в двойном слове

Примеры: I 1.0 - входной бит с номером 0 в байте номер1, Q 16.4 - выходной бит с номером 4 в байте номер16.

Для переменных типа BYTE в качестве абсолютного адреса используется идентификатор адреса и номер байта, в котором содержится значение переменной. Идентификатор адреса дополнен символом B. Примеры: IB 2 - входной байт номер 2, QB 18 - выходной байт номер 18.

Переменные типа WORD состоят из двух байтов (слово). В качестве абсолютного адреса используется идентификатор адреса и номер младшего байта машинного слова, в котором содержится значение переменной. Идентификатор адреса дополнен символом W.

Примеры: IW 4 - входное слово номер 4, содержит байты 4 и 5; QW 20 - выходное слово номер 20, содержит байты 20 и 21.

Переменные типа DWORD состоят из четырех байтов (двойное слово). В качестве абсолютного адреса используется идентификатор адреса и номер младшего байта двойного слова, в котором содержится значение переменной. Идентификатор адреса дополнен символом D.

Примеры: ID 8 - входное двойное слово номер 8, содержит байты 8, 9, 10 и 11; QD 24 - выходное двойное слово номер 24; содержит байты 24, 25, 26 и 27.

Примеры адресации области данных в блоке данных:

DB 10.DBX 2.0 - бит данных 2.0 в блоке данных DB 10,

DB 11.DBB 14 - байт данных 14 в блоке данных DB 11,

DB 20.DBW 20 - слово данных 20 в блоке данных DB 20,

DB 22.DBD 10 - двойное слово данных 10 в блоке данных DB 22.

Дополнительную информацию по адресации областей данных Вы найдете в разделе "Адресация данных".

3.2.2. Косвенная адресация

Косвенная адресация (indirect addressing) позволяет рассчитывать адреса в области данных во время выполнения программы В STL различают следующие виды адресации.

- Косвенная адресация посредством памяти ("Memory-indirect-addressing"), например, IW [MD 200] означает, что адрес находится в двойном слове памяти.

- Косвенная внутризонная адресация посредством регистра ("Register-indirect area-internal addressing") Например, IW [AR1, P#2.0] означает, что адрес, находящийся в адресном регистре AR1, получает приращение на величину смещения (offset) P#2.0 при выполнении оператора.

- Косвенная межзонная адресация посредством регистра ("Register-indirect area-crossing addressing") Например, W [AR1, P#0.0] означает, что адрес (включая адресную область), находящийся в адресном регистре AR1, получает приращение на величину смещения (offset) P#0.0 при выполнении оператора.

Двойные слова адресной области для данных (DBD и DID), меркеров (MD) и временных локальных данных (LD) могут использоваться для хранения адресов при косвенной адресации посредством памяти. Косвенную адресацию посредством регистра можно применять с использованием двух адресных регистров: AR1 и AR2. Косвенная адресация подробно описана в главе "Косвенная адресация".

3.2.3. Символьная адресация переменных

Символьная адресация (symbolic addressing) использует имена (символы) вместо абсолютных адресов. Вы сами можете выбирать эти имена. Такое имя должно начинаться с буквы и может содержать до 24 символов. В STL не разрешено использовать ключевые слова в качестве имен (символов).

При символьной адресации абсолютному адресу должно быть назначено имя (символ). Символы различаются по месту назначения: глобальные символы действительны во всей программе, тогда как локальные символы действительны только в блоке, в разделе объявления переменных которого они описаны.

Вы можете назначить имена (**глобальные символы**) в таблице символов (symbol table) следующим объектам:

- блокам данных и кодовым блокам,
- входам, выходам, периферийным входам и периферийным выходам,
- меркерам, таймерам и счетчикам,
- пользовательским типам данных,
- таблицам переменных.

Глобальный символ может также содержать пробелы, специальные символы и национальные символы, такие как умляут. Исключения составляют символы 00_{hex}, FF_{hex} и кавычки ("). При использовании в программе имен со специальными символами Вы должны заключать имена в кавычки. В скомпилированном блоке программный редактор всегда отображает все глобальные символы в кавычках.

Имена локальных переменных (**локальные символы**) определяются в разделе объявления переменных соответствующего блока. Эти имена могут содержать только буквы, цифры и знак подчеркивания. Локальные символы являются действующими только внутри блока, в котором они описаны. Такие же символы (такие же имена переменных) могут быть применены в ином контексте (для обозначения совершенно иных объектов) в другом блоке. Редактор отображает локальные символы с впереди стоящим символом "#". Если редактор не может отличить локальный символ от адреса Вы должны вводить этот символ с впереди стоящим символом "#".

Локальные символы доступны только в базе данных программатора PG (в автономном [offline] каталоге *Blocks*). Если эта информация отсутствует при декомпиляции, то редактор вставляет символы замены (substitute symbol).

Если Вы используете символьные имена во время инкрементного программирования, то эти имена должны уже к этому времени быть присвоены абсолютным адресам. Вы можете ввести новые символические имена в таблицу символов во время инкрементного программирования и можете в дальнейшем использовать их в программе.

3.3. Программирование кодовых блоков на STL

3.3.1. Структура STL-выражения

STL-программа состоит из ряда отдельных выражений (statement). Выражение - это наименьшая самостоятельная единица пользовательской программы. Выражение содержит описание команды для CPU. На рисунке 3.5 показана общая структура STL-выражения.

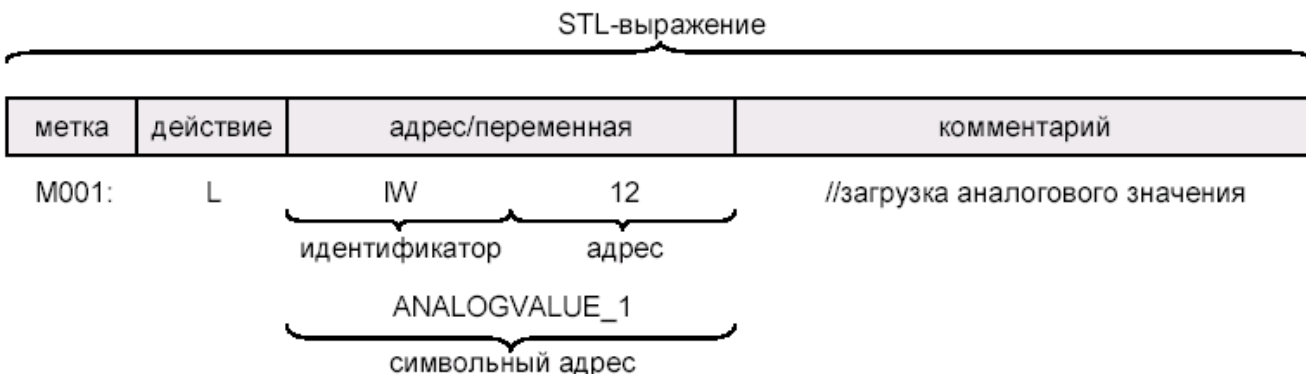


Рис. 3.5. Структура STL-выражения

Ниже перечислены компоненты STL-выражения.

- Метка (необязательный элемент) содержит до 4 символов, заканчивается двоеточием ":" (см. раздел "Функции перехода").
- Описание задания для CPU (код операции - оператор).
- Адрес - информация, необходимая для выполнения действия (например, абсолютный адрес IW12, символический адрес некоторой переменной ANALOGVALUE_1 или некоторая константа: W#16#F001 и т. д.). Некоторые операторы не требуют задания адреса.
- Комментарий (необязательный элемент) должен начинаться двумя косыми чертами "//" и продолжается до конца строки.

3.3.2. Инкрементное программирование кодовых блоков на STL

В разделе 2.3. "Создание S7-программ" Вы можете найти введение в основы создания S7-программы и использование программного редактора.

Процесс программирования блока начинается с его открытия одним из двух способов: либо двойным щелчком на блоке в окне проекта SIMATIC Manager, либо в редакторе с помощью выбора опций меню: *File -> Open (Файл -> Открыть)*. Если блок пока не существует, Вы должны его сначала создать одним из следующих путей.

- В левой половине окна проекта SIMATIC Manager выберите объект *Blocks (Блоки)*, создайте новый блок с помощью опций меню: *Insert->S7 Block->...*. В окне свойств (Properties) блока на вкладке "General - Part 1" ("Общие - часть 1") выберите номер блока и язык программирования "STL".
- Находясь в редакторе, с помощью опций меню: *File -> New (Файл -> Создать)* вызовите окно диалога с заголовком блока (номер блока, язык программирования, атрибуты блока). После закрытия диалогового окна Вы можете вводить программу этого блока.

Вы можете ввести информацию заголовка блока либо при создании блока, либо позднее, активизировав редактор, затем открыв блок и выбрав опции меню: *File -> Properties (Файл -> Свойства)*.

В редакторе программ язык программирования устанавливается на вкладке "Create Block" ("Создать блок") в диалоговом окне, открытом с помощью опций меню: *Options -> Customize (Опции -> Установка пользователя)*.

Таблица объявления переменных располагается выше окна программы. Если она не видна, то поместите курсор на верхней разделительной с окном программы линии, щелкните левой кнопкой мыши, когда курсор изменит свою форму, и "потащите" курсор. Вы увидите окно таблицы объявления переменных, в которой Вы должны дать описание для локальных переменных блока.

Существуют типы переменных, которые не могут использоваться в некоторых типах кодовых блоков. Если Вы не используете данный тип переменных, соответствующая строка должна остаться незаполненной.

Тип переменной (параметра блока) In (входные параметры) возможен в блоках FC, FB; Out (Выходные параметры) - в блоках FC, FB; In-out (Вх/вых параметры) - в блоках FC, FB; Stat (Статические локальные данные) - в блоках FB, Temp (Временные локальные данные) - в блоках OB, FC, FB.

Описание переменных состоит из имени, типа данных, значения по умолчанию (если есть) и комментария (не обязательный элемент). Не всем переменным может быть назначено значение по умолчанию (например, значения по умолчанию не могут присваиваться временным локальным данным). Более подробно значения, назначаемые по умолчанию для функций и функциональных блоков, описаны в разделе "Параметры блока".

Порядок описаний в кодовом блоке фиксирован, в то же время порядок следования типов переменных произволен. Вы можете экономно расходовать память, если будете группировать двоичные переменные в блоки по 8 или 16 штук, а переменные типа BYTE - парами. Редактор сохраняет новые переменные типов BOOL или BYTE, выделяя им байтовые участки памяти. Для всех остальных типов переменных выделяются участки памяти размером в 1 слово (начиная с байта с четным адресом).

В окне программы Вы увидите (в зависимости от установок редактора, принятых по умолчанию) поля для заголовка и комментария блока и, если это первый сегмент, то поля для заголовка и комментария сегмента, а также поле для ввода программы. В разделе программы кодового блока Вы можете управлять отображением комментариев и символов (имен) с помощью опций меню: *View -> Comment (Вид -> Комментарий)*, *View -> Symbolic Representation (Вид -> Представление символов)*, *View -> Symbol Information (Вид -> Информация о символе)*. Также Вы можете менять размер отображения с помощью опций меню: *View -> Zoom In (Вид -> Увеличить масштаб)*, *View -> Zoom Out (Вид -> Уменьшить масштаб)*, *View -> Zoom Factor (Вид -> Коэффициент масштабирования)*.

Вы можете разделить STL-программу на сегменты. Редактор нумерует сегменты автоматически, начиная с 1. Вы можете дать каждому сегменту заголовок сегмента (*network title*) и комментарий сегмента (*network comment*). Во время редактирования Вы можете выбирать каждый сегмент непосредственно с помощью опций меню: *Edit -> Go To -> ... (Правка -> Перейти к -> ...)*. Сегментирование не является обязательным приемом.

Для начала ввода программногo кода щелкните один раз кнопкой мыши ниже поля для комментария сегмента или, если Вы установили режим отображения "Display with Comments" ("Отображать с комментариями"), тогда щелкните один раз кнопкой мыши ниже выделенного тенью поля для комментария. Перед Вами пустая рамка окна. Здесь Вы можете вводить программу в любом месте внутри этого окна. Отделите оператор (OP-code) от адреса (операнд) одним или несколькими пробелами или одним шагом табуляции. После адреса в этой же строке Вы можете ввести две косых черты, после которых введите комментарий к выражению. Закончите выражение, нажав клавишу <Enter>.

Новый сегмент можно запрограммировать, выбрав опции меню: *Insert -> Network (Вставка -> Сегмент)*. При этом редактор вставляет пустой сегмент после выбранного сегмента. Если необходимо использовать символьные имена при инкрементном программировании, эти имена к моменту их использования в программе должны быть уже назначены абсолютным адресам.

Вы можете вызывать таблицу символов для выбора из нее символьных имен с помощью опций меню: *Insert -> Symbol (Вставка -> Символ)*. После вызова таблицы символов требуемый символ переносится в программу после щелчка на нем кнопкой мыши.

При инкрементном программировании Вы также можете вносить новые символьные имена в таблицу символов или корректировать имена, ранее в нее внесенные. Вы можете вызывать таблицу символов целиком с помощью опций меню: *Option -> Symbol Table (Опции -> Таблица символов)*, также Вы можете вызывать одну строку из таблицы символов для редактирования с помощью опций меню: *Edit -> Symbol (Правка -> Символ)*. После редактирования или ввода нового символьного имени Вы можете использовать его, продолжив ввод своей программы.

Нет необходимости завершать блок специальным выражением. Тем не менее, Вы можете запрограммировать последний "пустой" сегмент с заголовком "Block End" ("Конец блока"), облегчая тем самым зрительное восприятие программы (это может быть полезно в случае особо длинных

блоков). Если с помощью редактора открывается ранее скомпилированный блок, этот блок "декомпилируется", т.е. для него генерируется STL-код. Для этого редактор использует разделы программы в базе данных программатора PG, которые не совсем соответствуют программе, например, в плане символов, комментариев и меток перехода. Если нужная информация в базе данных программатора PG отсутствует во время декомпиляции программы, редактор использует символы-заместители (substitute symbols).

Вы можете в редакторе создать новые блоки или открыть и отредактировать существующие без необходимости перехода в утилиту SIMATIC Manager.

В отдельном подокне редактора отображается **каталог элементов программы**. Если он не видим, активизируйте его с помощью опций меню: *View -> Catalog (Bild -> Katalog)*.

Каталог элементов программы располагается в своем собственном подокне в левой части окна редактора. Каталог можно убрать с экрана, если дважды щелкнуть кнопкой манипулятора "мышь" на его заголовке. Каталог элементов программы поддерживается при программировании на языках LAD и FBD, обеспечивая доступные графические элементы. При программировании на языке STL этот каталог показывает только блоки, которые находятся в автономном (offline) каталоге *Blocks*, а также уже запрограммированные экземпляры мультитекстурных FB и доступные библиотеки.

3.4. Переменные и константы

3.4.1 Общие замечания по поводу переменных

Переменная - это величина особого формата (см. рис. 3.6). Простые переменные состоят из адреса

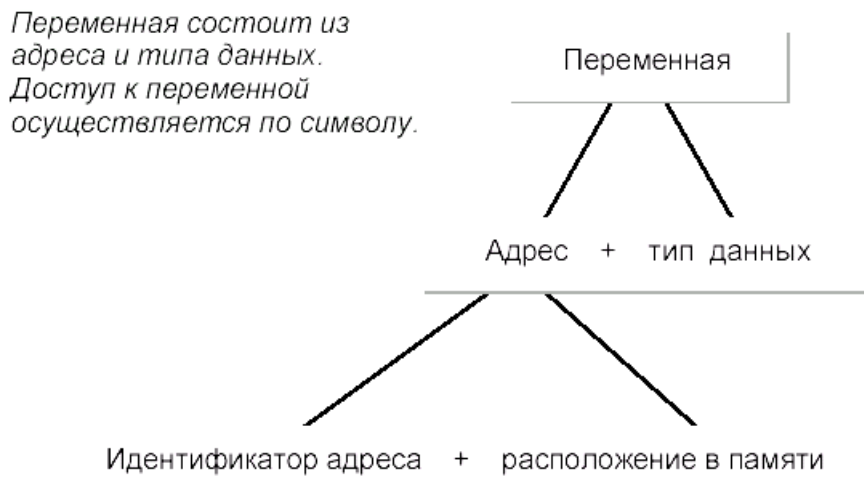


Рис. 3.6. Структура переменной.

(например, адрес входа I 5.2) и типа данных (например, BOOL для дискретного [двоичного] значения). Адрес, в свою очередь, содержит идентификатор адреса (такой как I для входа) и указание абсолютного месторасположения (такое, например, как 5.2, что означает 2-ой бит 5-ого байта). Вы можете также использовать для доступа к адресу или переменной символьное имя, назначив для этого адреса имя (символ) в таблице символов (Symbol Table).

Бит данных типа BOOL адресуется как *двоичный адрес (binary address)* или *двоичный операнд (binary operand)*. Адреса, содержащие один, два или четыре бита или переменные соответствующих типов называются *численными операндами (digital operand)*.

Переменные, объявленные внутри блока, являются внутриблочными (local - локальными) переменными. К ним относятся параметры блока, статические и временные локальные данные и даже адреса данных в блоках глобальных данных. Если локальные переменные являются переменными простого типа, они также могут быть доступны как операнды (например, статические локальные данные - как DI операнды, временные локальные данные - как L операнды, и данные в блоках глобальных данных - как DB операнды). Тем не менее, локальные переменные могут быть сложных типов (например, такие переменные как структура или массив). Переменные этих типов требуют для размещения памяти больше, чем 32 бита, так что они не могут быть, например, загружены в аккумулятор. И по этой же причине они не могут быть адресованы с помощью обычного ("normal") STL-выражения. Существуют специальные

функции для обработки таких переменных, такие как IEC-функции, которые поставляются в виде стандартной библиотеки с ПО STEP 7 (Вы можете создавать переменные сложных типов в параметрах блоков такого же типа).

Если переменные сложного типа содержат компоненты простого типа, то эти компоненты могут рассматриваться, как если бы они были отдельными переменными (например, Вы можете загрузить в аккумулятор элемент массива, состоящего из 30 значений целого (INT) типа, и в дальнейшем обрабатывать его).

Константы используются для задания переменным значений. Константа имеет особый префикс в зависимости от типа данных, к которому она принадлежит.

Тип данных обуславливает характеристики данных, особенно это касается представления содержания переменной и диапазона допустимых для нее значений. STEP 7 предусматривает предопределение типов данных. Вы можете комбинировать типы данных, формируя пользовательские типы данных (User Data Type - UDT). Типы данных доступны в базовом пакете (Global Basis) и могут использоваться в каждом блоке. Ниже в таблице 3.2 приведен обзор типов данных в STEP 7.

Табл. 3.2.

Elementary Data Types (Простые типы данных)	Complex Data Types (Сложные типы данных)	User Data Types (Пользовательские типы данных)	Parametr Data Types (Типы данных для параметров)
BOOL, BYTE, CHAR, WORD, INT, DATE, DWORD, DINT, REAL, S5TIME, TIME, TOD	DT, STRING, ARRAY, STRUCT	UDT, блоки глобальных данных, экземпляры (Instances)	TIMER, COUNTER, BLOCK_DB, BLOCK_SDB, BLOCK_FC, BLOCK_FB, POINTER, ANY
Типы данных с объемом не более одного двойного слова (32 бита)	Типы, которые могут содержать данные объемом более одного двойного слова (DT, STRING) или состоящие из нескольких компонентов	Структуры или области данных, которые могут быть адресованы по имени	Параметры блока
Могут распределяться для операндов абсолютной или символьной адресацией	Могут распределяться только для переменных с символьной адресацией		Могут распределяться только для параметров блоков (только с символьной адресацией)
Разрешенные во всех адресных областях	Разрешенные в блоках данных (как глобальные данные и экземплярные данные), как временные локальные данные и как параметры блоков		Разрешенные в связи с параметрами блоков

3.4.2. Простые типы данных

Переменные простых типов данных могут быть отредактированы непосредственно в редакторе STL-программ, так как их значения могут занимать от одного до 32 битов в памяти. Переменные простых типов могут быть предопределены фиксированными значениями (константами) на этапе объявления переменных. Для многих типов данных существует больше одной формы для записи константы и все эти записи одинаково правомочны в употреблении (например, можно использовать форму записи TIME# или форму записи T#).

Язык STL не накладывает ограничений на обработку (на операторы) типов данных (за исключением различий для двоичных [binary] и численных [digital] операндов). Функции сравнения, такие например, как сравнение содержимого аккумуляторов, не зависят от типа переменных, содержащихся в аккумуляторах.

3.4.3. Сложные типы данных

Вы можете использовать сложные типы данных (табл. 3.2) для работы с переменными в блоках данных или в L-стеке, а также для работы с параметрами блока. Переменные сложных ("complex") типов, которые назначаются параметрам блоков, могут быть только полными переменными; отдельные части переменных сложных типов не могут быть обработаны с помощью обычных операторов. Тем не менее с помощью прямого доступа к переменным и с помощью косвенной адресации STL обеспечивает способ управления переменными, если известна их внутренняя структура. Кроме того существуют IEC-функции, с помощью которых можно обрабатывать переменные DT и STRING (например, объединение двух символьных строк в одну). IEC-функции являются составной частью системы STEP 7; Вы можете найти их в стандартной библиотеке "Standard Library" в разделе "IEC Function Blocks". IEC-функции могут быть использованы в любом языке программирования. Длина переменной DT является

фиксированной; Вы можете сами задавать длину переменных STRING, ARRAY и STRUCT при их определении.

Строка символов STRING может содержать до 254 символов и резервирует в памяти на 2 байта больше, чем число символов в строке. Массив может иметь до 65536 элементов для каждого из индексов (т.е. от -32768 до 32767).

3.4.4. Параметрические типы

Параметрические типы - это типы данных для параметров блоков (см. табл. 3.2). Размеры данных в таблице 3.3 соответствуют области памяти, требуемой для параметров блока (для случая функциональных блоков). Используйте также TIMER и COUNTER в таблице символов как типы данных для таймеров и счетчиков).

Таблица 3.3.

Параметрический тип	Описание	Примеры фактических адресов
TIMER	Таймер 16 бит	T 15 (или символ)
COUNTER	Счетчик 16 бит	C 16 (или символ)
BLOCK_FC	Функция 16 бит	FC 17 или символ
BLOCK_FB	Функциональный блок 16 бит	FB 18 или символ
BLOCK_DB	Блок данных 16 бит	DB 19 или символ
BLOCK_SDB	Блок системных данных 16 бит	SDB 100 или символ
POINTER	DB-указатель 48 бит	P#M10.0 (указатель) P#DB20.DBX22.2 (указатель)
ANY	ANY-указатель 80 бит	P#DB10.DBX0.0 WORD 20

Примеры использования параметрических типов в программе:

```
SI #Contact_time      // Contact time: TIMER
LC #No_Comp_Parts     // NoCompParts: COUNTER
UC #Recall             // Recall: BLOCK_FB
L P##Measure          // Measure: POINTER
L P##Meas_Values      // Meas_Values: ANY
```

4. Базовые функции

Базовые функции позволяют программировать контроллер как систему управления на контакторах и реле.

4.1. Битовые операции

4.1.1. Служебные биты

В SIMATIC S7 для битовых операций используются различные специальные битовые регистры, входящие в слово статуса CPU, и статусы битов дискретных сигналов.

Состояние (статус) бита эквивалентно состоянию его сигнала и может иметь значения "0" или "1". Состояние сигнала имеет значение "1" при наличии напряжения на входе в заданном для соответствующего сигнального модуля диапазоне; с другой стороны, если напряжение на входе модуля находится в другом диапазоне (в зависимости от типа модуля), то состояние сигнала входа соответствует "0".

Результат логической операции (RLO) - это состояние сигнала (битовый регистр), которое в дальнейшем используется для обработки двоичных сигналов. Результат логической операции формируется и модифицируется с помощью операторов проверки (check statement).

Если RLO имеет значение "1", то это означает, что условие двоичной логической операции выполнено, если RLO имеет значение "0", то это означает, что условие двоичной логической операции не выполнено. Биты устанавливаются или сбрасываются в соответствии с результатом логической операции.

Условные операторы (condition statement) - это такие операторы, выполнение которых зависит от результата логической операции RLO. Эти операторы включают операции присвоения (назначения [assign]), установки и сброса двоичных разрядов, а также запуск таймеров и счетчиков и т.п.

Условные операторы (за редким исключением) выполняются, когда результат логической операции RLO имеет состояние "1", и не выполняются, когда RLO имеет состояние "0". Условные операторы (за редким исключением) не влияют на результат логической операции RLO, и, таким образом, RLO на протяжении последовательного выполнения нескольких операторов остается неизменным.

Среди битов слова статуса CPU следует выделить его нулевой бит, называемый битом первичного опроса (обозначается как **/FC**). Он устанавливается в «1» любым оператором проверки и сбрасывается в «0» любым условным оператором.

Выражение, содержащее **оператор проверки**, инициирует проверку состояния бита. В то же время это выражение содержит правило логической операции - алгоритм, согласно которому результат проверки состояния бита будет сравниваться с сохраненным в процессоре результатом предыдущей логической операции и связываться (link) с ним в соответствии с правилом AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Затем CPU сохраняет результат данной логической операции как новое значение RLO.

Например, выражение «A I 17.1» проверяет вход I 17.1 на состояние "1" и связывает с RLO по логике AND (И). Выражение «ON M 20.5» проверяет меркер M 20.5 на состояние "0" и связывает с RLO по правилу логической операции OR (ИЛИ).

Первый оператор проверки, следующий за условным оператором, называется **"первичный опрос"** ("first check"). Он имеет особенное значение, потому что CPU имеет прямой доступ к результату этого оператора, как результату логической операции. Таким образом "старое значение" результата логической операции теряется. Первичный опрос всегда соответствует началу логической операции. Правило (алгоритм) первичного опроса (И, ИЛИ, Исключающее ИЛИ) не играет при этом никакой роли.

Результат логической операции формируется с помощью двоичной логической операции (первичного опроса или оператора проверки). На рис. 4.1 показана общая схема её выполнения в зависимости от /FC.

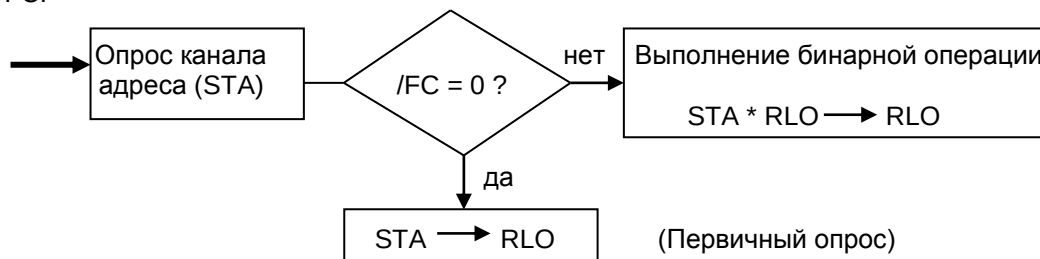


Рис. 4.1. Схема выполнения логической операции (* - шаблон логической операции).

Входной модуль выбирает датчик посредством адреса, например, датчик на входе I 1.2. CPU проверяет состояние сигнала (статус) датчика связывает его на входе блока логической операции с результатом логической операции, сохраненным в предыдущий раз. Результат текущей логической операции (RLO) запоминается и хранится как "новый результат логической операции". Затем CPU обрабатывает следующее выражение программы, например, обеспечивающее сохранение результата логической операции в специальной ячейке памяти.

Строго говоря, CPU не связывает состояние сигнала проверенного бита, а скорее формирует результат проверки: в случаях проверки на состояние сигнала "1" результат проверки идентичен состоянию проверяемого бита; в случаях проверки на состояние сигнала "0" результат проверки инвертируется относительно состояния проверяемого бита.

Так же, как можно выделить последовательный шаг в системе управления, так и в логической системе управления можно выделить **логический шаг** (logic step). В каждом таком логическом шаге формируются и оцениваются (т.е. подвергается последующей обработке) результаты последовательности логических операций. Логический шаг состоит из выражений с операторами проверки (также называемыми операторами сканирования) между условными операторами. Таким образом перед началом логического шага и сразу после него бит первичного опроса равен нулю, а для всех операций шага - единице. Первый оператор проверки, следующий за условным, с него и начинается логический шаг. В нижеследующей схеме логической системы управления выделены два логических шага:

```

= Q 4.0      //условный оператор, /FC = 0
// Первый логический шаг
A I 2.0      первичный опрос, /FC = 1
A I 2.1      оператор проверки, /FC = 1
A I 1.7      оператор проверки, конец логического шага, /FC = 1
= Q 5.1      условный оператор, /FC = 0
= Q 4.3      условный оператор, /FC = 0
// Второй логический шаг
  
```

- O I 2.6 первая проверка, /FC = 1
- O I 2.5 оператор проверки
- ...

Правило (логический алгоритм) первичного опроса не имеет значения, так как результат этой проверки берется непосредственно как результат логической операции. В целях грамотного программирования правило первичного опроса должно быть идентично требуемой функции. Например, последовательность операторов

```
.
= Q 15.3
// первая AND функция
O I 18.5
A I 21.7
= Q 15.4
// вторая AND функция
A I 18.4
A I 21.6
= Q 15.5
.
```

представляет собой две AND-функции, из которых вторая функция (в которой обе проверки имеют логику AND) запрограммирована более предпочтительным способом.

В случае единичных операторов проверки, как например, в следующем фрагменте:

```
.
= Q 10.0
A I 20.1 // присвоение значения RLO
= Q 10.1 // в итоге: I 20.1 -> Q 10.1
.
```

функция AND более предпочтительна, чем какая-нибудь другая.

4.1.2. Элементарные двоичные логические операции

В языке программирования STL используются элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Эти функции связаны с проверкой сигнала на состояние "1" и "0".

```
A адрес бита // проверка сигнала на состояние "1" и комбинирование в соответствии с функцией AND (И)
AN адрес бита // проверка сигнала на состояние "0" и комбинирование в соответствии с функцией AND (И)
O адрес бита // проверка сигнала на состояние "1" и комбинирование в соответствии с функцией OR (ИЛИ)
ON адрес бита // проверка сигнала на состояние "0" и комбинирование в соответствии с функцией OR (ИЛИ)
X адрес бита // проверка сигнала на состояние "1" и комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)
XN адрес бита // проверка сигнала на состояние "0" и комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)
```

Проверка сигнала на состояние "1" дает результат проверки "1", если состояние сигнала бита соответствует "1". Проверка сигнала на состояние "0" дает результат проверки "1", если состояние сигнала бита соответствует "0".

Если опрос не первичный, то CPU комбинирует результат проверки бита с текущим результатом логической операции (RLO) в соответствии с логической функцией и формирует новое значение RLO; иначе, результат проверки вводится в RLO-буфер без выполнения логической операции.

Число двоичных функций и диапазон их действия теоретически произвольны. На практике, однако, ограничения на использование этих функций накладываются из-за конечных размеров блока и рабочей памяти CPU.

Функция AND (И) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) равны "1". Если функция AND встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы все промежуточные RLO были равны "1", иначе общий результат будет равен "0".

Функция OR (ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если хоть один из этих сигналов (один из результатов проверки) равен "1". Если функция OR встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы хоть один промежуточный RLO был равен "1". Если все промежуточные RLO равны "0", то общий результат будет равен "0".

Функция Exclusive OR (Исключающее ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) имеют разные значения; с другой стороны, RLO равен "0", если оба эти сигнала имеют одинаковое значение.

Элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ), описываемые в предыдущих разделах данной главы, рассматривались как входные модули с нормальноразомкнутыми контактами на входах (т.е. датчик имеет нормальноразомкнутые контакты, которые замыкаются при активации датчика, и при этом датчик возвращает значение сигнала, равное "1"). При выполнении функции управления, однако, не всегда возможно использовать только датчики, имеющие нормальноразомкнутые контакты. Во многих случаях, например, в случае "закороченных" цепей (имеющих короткое замыкание), использование нормальнозамкнутых контактов совершенно необходимо (нормальнозамкнутые контакты датчика обеспечивают возвращение сигнала, равного "0", при активации датчика).

Если датчик, подключенный ко входу, имеет нормальноразомкнутые контакты, то при активации датчика на входе возникает значение сигнала, равное "1". Если датчик, подключенный ко входу, имеет нормальнозамкнутые контакты, то в неактивном состоянии датчика на входе сохраняется значение сигнала, равное "1". CPU "не знает" к какому типу относится датчик на том или ином входе (нормальнозамкнутый или нормальноразомкнутый). Он может только различить состояния сигналов "1" или "0". При разработке программы, следовательно, необходимо учитывать тип используемого датчика.

Перед вводом программы Вы должны определиться с типом используемого датчика (с нормальнозамкнутыми или с нормальноразомкнутыми контактами). Это необходимо, потому что в отдельных частях программы учитывается состояние датчиков ("Sensor activated" [Датчик активирован], "Sensor not activated" [Датчик не активен]), а, следовательно, Вы должны проверять вход на состояние сигнала "1" или на состояние "0", в зависимости от типа используемого датчика.

Оператор NOT (НЕ) инвертирует результат логической операции. Вы можете использовать оператор NOT (НЕ), например, для инвертирования результата выполнения функции AND (И)

4.1.3. Сложные двоичные логические операции

Двоичные логические функции могут быть объединены, например, функции AND (И) и OR (ИЛИ) могут стоять в программе в любом порядке. Если такие функции стоят в программе в произвольном порядке, нелегко разобраться в обработке их CPU. Поэтому лучше использовать для иллюстрации решения задачи, например, программирование с помощью функциональных блок-схем, чем программирование на STL.

При программировании сложных двоичных логических операций STL одинаково рассматривает операторы OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ) как операторы с одинаковым приоритетом. Оператор AND (И) имеет более высокий приоритет и выполняется "перед" операторами OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ).

Чтобы функции выполнялись в требуемом порядке CPU иногда должен временно сохранять значение функции (результат логической операции RLO, рассчитанный в некоторой точке программы). Для этой цели (для временного хранения результата) могут использоваться вложенные выражения (вложенные операторы). Как и в случае используемых в булевой алгебре записей вложенные операторы обеспечивают выполнение одних функций раньше других. Вложенные выражения (операторы) могут также включать в себя функцию OR (ИЛИ).

Язык программирования STL позволяет использование следующих двоичных вложенных выражений (вложенных операторов):

- O - функция OR (ИЛИ) для функций AND (И),
- A(- открывающая скобка с функцией AND (И),
- O(- открывающая скобка с функцией OR (ИЛИ),
- X(- открывающая скобка с функцией Exclusive OR (Исключающее ИЛИ),
- AN(- открывающая скобка с функцией NOT-AND (НЕ-И),
- ON(- открывающая скобка с функцией NOT-OR (НЕ-ИЛИ),
- XN(- открывающая скобка с функцией NOT-Exclusive OR (НЕ-Исключающее ИЛИ),
-) - закрывающая скобка.

Правило логики для выражения с открывающей скобкой показывает, как результат вложенного выражения должен быть связан с текущим значением RLO в момент обработки закрывающей скобки. До этой логической операции результат выполнения вложенного выражения инвертируется, если присутствует символ операции инвертирования.

4.1.4. Обработка вложенных выражений (вложенных операторов)

В языке программирования STL двоичные вложенные выражения используются для определения порядка выполнения двоичных логических операций. В процессе выполнения программы CPU первыми обрабатывает выражения, заключенные в скобки, то есть до выполнения выражений, находящихся за скобками.

Когда CPU встречает открывающую скобку, запоминает текущее значение RLO и затем обрабатывает выражение в скобках (вложенное выражение), когда CPU встречает закрывающую скобку, он связывает значение RLO для вложенного выражения с ранее запомненным значением RLO в соответствии с функцией, определенной при открывающей скобке.

Оператор проверки, следующий за открытой скобкой, всегда является первичным опросом, т.к. CPU всегда создает новый результат логической операции RLO для вложенного выражения. Оператор проверки, следующий за закрытой скобкой, никогда не является первичным опросом, т.к. первой инструкцией является вложенное выражение. CPU обрабатывает значение RLO для вложенного выражения как результат первичного опроса.

Вложенные выражения могут включать в себя другие вложенные выражения. Глубина вложения равна 7, что означает, что Вы можете 7 раз включать новые вложенные выражения в выражения, которые уже сами по себе являются вложенными, не завершая последних. Обработка внутренних скобок ведется, как описано выше.

При обработке вложенных выражений в CPU заполняется "стек вложения" ("nesting stack") в порядке обработки вложенных функций. В данном стеке хранится следующая информация:

- результат логической операции (RLO) предыдущих скобок;
- двоичный результат (BR "binary result") предыдущих скобок;
- бит состояния (OR) (показывающий, было ли уже выполнено условие функции OR [ИЛИ]);
- функция вложения (для запоминания функции, с которой необходимо связать результат вложенного выражения).

CPU устанавливает двоичный результат (BR) после закрывающей скобки в состояние, которое имело место к началу обработки вложенного выражения. Во вложенных выражениях Вы можете использовать не только двоичные логические операторы, но и все остальные выражения языка программирования STL. Однако, от программиста требуется внимание при завершении сложного вложенного выражения закрывающей скобкой. Так, возможно, например, во вложенном выражении запрограммировать несколько логических шагов, или операций с памятью, или функций сравнения.

4.1.5. Объединение AND-функций (И) в операторе OR (ИЛИ)

Эти логические операции, представляющие собой комбинации функций OR (ИЛИ) и AND (И), могут быть записаны в булевой алгебре без использования скобок. Здесь действует правило, согласно которому функции AND выполняются в первую очередь. Затем результаты, если есть функция OR, обрабатываются (link - связываются) этой функцией.

Пример:

```
A   Input0
A   Input1
O
A   Input2
A   Input3
=   Output8
```

В данном примере единственный оператор O (соответствующий функции ИЛИ) находится между двумя функциями И. В примере Output8 устанавливается, если { Input0 И Input1 } ИЛИ { Input2 И Input3 } установлены (имеют значение "1").

4.1.6. Объединение ИЛИ и Исключающее ИЛИ в операторе И

Логические операции, представляющие собой комбинации логических функций, должны быть записаны в булевой алгебре с использованием скобок, с помощью которых указывается, какие функции выполняются в первую очередь.

Пример:

```
A(
O   Input0
O   Input1
)
A(
O   Input2
```

```

O   Input3
)
=   Output10

```

В примере Output10 устанавливается, если { Input0 ИЛИ Input1} И { Input2 ИЛИ Input3} установлены (имеют значение "1").

Точно также как Вы можете проверить бит на состояние сигнала "0" (по сути инвертировать состояние бита), Вы также можете инвертировать вложенное выражение (точнее - инвертировать результат этого выражения). Признаком инвертирования результата операции является наличие дополнительного символа N в выражении открывающей скобки.

Пример:

```

AN(
O   Input0
O   Input1
)
AN(
X   Input2
X   Input3
)
=   Output16

```

В примере Output16 устанавливается, если ни в одной из скобок - ни в {Input0 ИЛИ Input1}, ни в { Input2 ИЛИ Input3} - не выполняется условие вложенных функций.

Второй способ инвертирования вложенных выражений возможен с помощью использования выражения (оператора) NOT. Оператор NOT, записанный перед закрывающей скобкой, инвертирует результат вложенного выражения перед последующей обработкой.

Пример:

```

A(
O   Input0
O   Input1
NOT
)
A(
X   Input2
X   Input3
NOT
)
=   Output17

```

В данном примере имеет место такая же логическая операция, как и в предыдущем примере.

Инструкции битовой логики могут также использоваться для прямого контроля слова состояния при использовании следующих **адресов**: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Пример:

```

L MW20      // Загрузка в аккумулятор значения первого аргумента из слова памяти
L Prior     // Загрузка в аккумулятор значения второго аргумента (переменной)
+I          // Сложение загруженных значений как 16-тибитовых целых
A ==0       // Единичный опрос битов состояния на нулевое значение результата сложения
= M10.0     // Присвоение результата опроса биту памяти
A <>0       // Единичный опрос на ненулевое значение результата сложения
= M10.1     // Присвоение результата опроса биту памяти

```

4.2. Операции с памятью (memory functions)

В данном разделе рассматриваются операции с памятью, применяемые в языке программирования STL. К операциям с памятью относятся функции Присвоение для "динамического" управления битами, а также функции Установка бита и Сброс бита для "статического" управления битами. Также к операциям с памятью относится функция проверки наличия фронта сигнала.

Операции с памятью используются в сочетании с двоичными логическими операциями, чтобы воздействовать на значения сигналов (состояния) битов с использованием "результата логической операции" RLO, который генерируется CPU.

Вы можете использовать функции работы с памятью для управления всеми битовыми адресами: адресами области отображения входов/выходов процесса, адресами меркеров, адресами глобальных данных, адресами статических и временных локальных данных.

4.4.1. Функция Присвоение

Синтаксис: = *Bit*

Функция присваивает биту результат логической операции (RLO). Выражение, содержащее символ присвоения "=", присваивает RLO, содержащийся в CPU, биту с адресом *Bit*, указанному в операнде. Если результат логической операции имеет значение "1", то бит устанавливается; если результат логической операции имеет значение "0", то бит сбрасывается.

Если необходимо установить бит, в то время когда RLO имеет значение "0", то значение RLO должно быть сначала инвертировано с помощью оператора NOT до момента выполнения функции присвоения.

Также допускается выполнять одновременное присваивание результата логической операции различным битам, запрограммировав подряд несколько операторов присваивания, так как инструкции присваивания не влияют на RLO. CPU не генерирует новый результат логической операции RLO до появления следующего оператора проверки.

Если необходимо проверить состояние сигнала выхода в другой логической операции просто проверьте этот выход с помощью соответствующей функции проверки.

4.4.2. Функции Установка бита) и Сброс бита

Синтаксис: S *Bit*

Функция устанавливает бит с адресом *Bit*, в случае если RLO равен "1", а

R *Bit*

сбрасывает бит с адресом *Bit*, в случае если RLO равен "1".

Функция S устанавливает бит, а функция R сбрасывает бит только в случае, если RLO имеет значение "1". Если результат логической операции имеет значение "0", то инструкции S и R не изменяют состояния бита, определенного как операнд в этих инструкциях.

Допускается использование нескольких операций с памятью одновременно и в любой комбинации, а также совместно с функциями присвоения с использованием одного и того же результата логической операции. Просто запишите в программе подряд несколько операторов установки или сброса для соответствующих битов. Во время действия функций присвоения, установки и сброса RLO не изменяет своего значения. При этом CPU не генерирует новый результат логической операции до появления следующего оператора проверки. Вы можете также использовать оператор NOT для инвертирования значения RLO в одном ряду с операциями с памятью.

Для большей ясности и лучшей читаемости программы рекомендуется использовать операторы S или R парами для определенного бита и только один раз.

4.4.3. Функция «Проверка наличия фронта сигнала»

Синтаксис: FP *Bit* // Функция проверки наличия положительного или переднего (возрастающего) фронта сигнала,

FN *Bit* // Функция проверки наличия отрицательного или заднего (убывающего) фронта сигнала.

CPU определяет положительный (возрастающий) фронт сигнала, когда до начала выполнения функции результат логической операции изменяется от уровня "0" в предыдущем цикле к уровню "1" в текущем цикле. CPU определяет отрицательный (убывающий) фронт сигнала, когда до начала выполнения функции результат логической операции изменяется от уровня "1" к уровню "0".

Бит, указанный в операторе проверки фронта, сохраняет предыдущее состояние RLO. Результат выполнения функции проверки фронта отображается с помощью RLO = 1.

В каждом цикле, состояние бита RLO сравнивается с его значением в предыдущем цикле для определения появления фронта RLO. Состояние RLO в предыдущем цикле для этого сохраняется в бите памяти (*Bit*) для дальнейшего сравнения. При появлении нужного отличия между текущим и предыдущим состоянием RLO на выходе выдается состояние "1" (выделение фронта), т.е. бит RLO после выполнения этой инструкции принимает значение "1". В противном случае RLO сбрасывается.

Примечание: Нельзя использовать в качестве бита памяти локальные данные, так как они актуальны только на момент выполнения текущего блока.

Функция проверки наличия переднего или заднего фронта сигнала сбрасывает бит **OR** слова статуса и устанавливает бит **/FC**.

В логических переключающих схемах эквивалентом функции проверки наличия фронта сигнала является "контактный формирователь импульса" ("pulse contact element"). При включении реле этот

формирователь импульса генерирует импульс, свидетельствующий о наличии возрастающего фронта сигнала. При выключении реле этот формирователь импульса генерирует импульс, свидетельствующий о наличии убывающего фронта сигнала. К биту, указанному как операнд в функции проверки наличия фронта сигнала, обращаются как к "меркеру фронта" ("edge memory bit") (хотя фактически этот бит может быть и не из области меркеров). Тем не менее, он должен быть таким битом, состояние сигнала которого может быть проверено в любой момент в последующих циклах сканирования программы, и который не используется в программе каким-либо другим образом.

Битом в функции проверки наличия фронта сигнала может быть меркер, бит из блока глобальных данных или бит из статических локальных данных в функциональных блоках. (В дальнейшем рассматриваемый бит именуется для определенности "меркером фронта").

Непосредственно после выполнения функции обнаружения фронта сигнала Вы можете использовать полученное значение RLO с помощью функций проверки. Проверьте реакцию функции обнаружения фронта сигнала после включения CPU. Для обнаружения фронта сигнала нужно, чтобы RLO до выполнения функции обнаружения фронта и состояние меркера фронта были одинаковыми. При определенных обстоятельствах меркер фронта должен быть сброшен при запуске (в зависимости от требуемой реакции функции проверки наличия фронта, а также от используемого бита).

4.4.4. Проверка наличия фронта в двоичной логической операции

Проверка наличия фронта в двоичной логической операции может служить для решения практической задачи, если Вы используете сигнал, полученный в результате проверки ("импульс") для управления таймером, счетчиком или операцией с памятью. Двоичные операции проверки могут располагаться между операцией проверки наличия фронта и запускаемой функцией, например:

```
O   Input3
O   Input4
FP  EMembit2
A   Input5
S   Output15
A   Input6
FN  EMembit3
R   Output15
```

В примере выход *Output15* устанавливается в момент, когда выполняется OR- условие (когда бит в OR-выражении переходит от состояния "0" к состоянию "1") и вход *Input5* установлен (равен "1"). Выход *Output15* сбрасывается в момент, когда приходит отрицательный фронт на вход *Input6*.

Функция проверки наличия фронта сигнала является "первичным опросом" ("first check"), так как результат логической операции RLO, который генерируется функцией, может быть использован для последующей обработки. Это также означает, что логическая операция до момента начала проверки фронта считается "завершенной". Функция проверки наличия фронта сигнала не влияет на обработку вложенных инструкций.

4.4.5 «Двоичный делитель»

В качестве примера использования команды FP рассмотрим программу «Двоичный делитель». Она имеет один вход и один выход. Если сигнал на входе двоичного делителя меняет свое состояние, например, с состояния "0" на состояние "1", то выходной сигнал также будет менять свое состояние. В примере выходной сигнал сохраняет состояние до появления нового положительного фронта сигнала на входе делителя. Только при этом условии выходной сигнал делителя изменяет свое состояние. Это означает, что в этом случае частота изменения выходного сигнала составляет половину частоты входного сигнала.

В нижеследующей программе для первого варианта приняты обозначения: Input - вход, Output - выход, EMembit - меркер фронта, PMembit - меркер импульса.

```
A   Input_1
FP  EMembit_1
=   PMembit_1
A   PMembit_1
AN  Output_1
S   Output_1
R   PMembit_1
A   PMembit_1
A   Output_1
```

4.3. Функции пересылки данных

В данном разделе рассматриваются функции языка программирования STL, с помощью которых выполняются операции обмена данными с помощью аккумуляторов (регистров). К ним относятся следующие функции (команды):

- L - функция загрузки данных в аккумулятор 1, которая используется для загрузки данных в аккумулятор для последующей обработки (выполнение функций обработки чисел, операций сравнения, арифметических операций и т.д.);
- T - функция выгрузки данных из аккумулятора 1 в память CPU, например, в область меркеров;
- ТАК - функция обмена значений аккумуляторов.

Функции L Вам также могут потребоваться для задания начальных значений для таймеров и счетчиков или для обработки текущих значений таймеров и счетчиков. Функции L и функции T также могут потребоваться для адресации модулей с использованием области данных пользователя.

Если для адресации модулей Вы используете область системных данных, Вы должны использовать системные функции для передачи записей данных. Эти системные функции также можно использовать для параметризации модулей. Системные функции SFC 20 (BLKMOV), SFC 81 (UBLKMOV) и SFC 21 (FILL) используются для копирования больших объемов информации в памяти или в заданные области данных.

4.3.1 Общие замечания по поводу операций загрузки и выгрузки данных

Функции загрузки данных в аккумулятор и функции выгрузки данных из аккумулятора позволяют производить обмен между различными областями памяти. Такой обмен данными не может производиться непосредственно, а только с использованием аккумулятора 1. Аккумулятор - это специальный регистр в процессоре, который выполняет функции промежуточного буфера. Во время обмена информацией направление, в котором происходит передача данных, указывается в используемой для передачи инструкции. Данные, направляемые из памяти в аккумулятор 1, называются *загружаемыми* (loading), тогда как данные, пересылаемые в обратном направлении, называются *выгружаемыми* (transferring, содержимое аккумулятора "выгружается" в область памяти).

Операции загрузки данных в аккумулятор и выгрузки данных из аккумулятора являются предопределенными операциями для выполнения *функций обработки чисел* (digital functions), с помощью которых осуществляется управление численными значениями (digital value) (например, операция сдвига или преобразования) или комбинирование двух чисел (например, операция сложения или сравнения). Для одновременной обработки двух численных значений требуются два промежуточных буфера. В роли таких буферов выступают аккумулятор 1 и аккумулятор 2.

Все CPU имеют такие специальные регистры. Кроме того, S7-400 CPU имеют два дополнительных промежуточных буфера - аккумулятор 3 и аккумулятор 4, которые используются преимущественно в арифметических операциях.

Несколько функций, называемых функциями аккумуляторов, используются для копирования содержимого одного аккумулятора в другой.

Функции загрузки пересылают информацию из системной памяти (system memory), рабочей памяти (work memory) и периферии (I/O) в аккумулятор 1, смещая при этом "старое" (точнее сказать, "текущее") значение аккумулятора 1 в аккумулятор 2. Функции обработки чисел позволяют управлять содержимым аккумулятора 1 или комбинировать численные значения, содержащиеся в аккумуляторах 1 и 2, с последующей записью результата в аккумулятор 1.

Функции аккумуляторов позволяют получить доступ к содержимому всех аккумуляторов. Источником для пересылки информации в системную память, рабочую память и в периферию может служить лишь только аккумулятор 1.

Каждый аккумулятор содержит 32 разряда, тогда как все области памяти имеют байтовую структуру. Обмен информацией между областями памяти и аккумулятором 1 может происходить побайтно, по 1 машинному слову и по 1 двойному машинному слову. Функции загрузки данных в аккумулятор и функции выгрузки могут быть применены со следующими адресными областями:

- области таймеров и счетчиков,
- слово состояния,
- области временных локальных данных,
- области адресов данных, длина и число блоков данных,
- адресные регистры и указатели,

- адреса переменных.

4.3.2 Функции загрузки данных в аккумулятор

Функция загрузки состоит из оператора L (код операции загрузки) и константы, переменной или адреса с идентификатором адреса, содержимое которого функция будет загружать в аккумулятор 1:

```
L +1200    // Константа (прямая адресация)
L IW 16    // Местоположение числа (абсолютная адресация)
L ActValue // Переменная (символьная адресация)
```

CPU выполняет функцию загрузки независимо от результата логической операции (RLO) и битов состояния. Функция загрузки не влияет на результат логической операции и не влияет на биты состояния.

Функция загрузки данных в аккумулятор влияет на содержимое аккумулятора 2. В то время как значение адреса, константы или переменной, определенное в операторе загрузки загружается в аккумулятор 1, текущее содержимое аккумулятора 1 пересылается в аккумулятор 2. Функция загрузки Load полностью пересылает содержимое аккумулятора accumulator 1 в аккумулятор accumulator 2. Предыдущее содержимое аккумулятора 2 при этом теряется.

При использовании S7-400 CPU функция загрузки данных не влияет на содержимое аккумуляторов 3 и 4.

Адрес числа, определенный в функции загрузки, может иметь размер байта (byte), слова (word) или двойного слова (double word) (см. рис. 4.2).

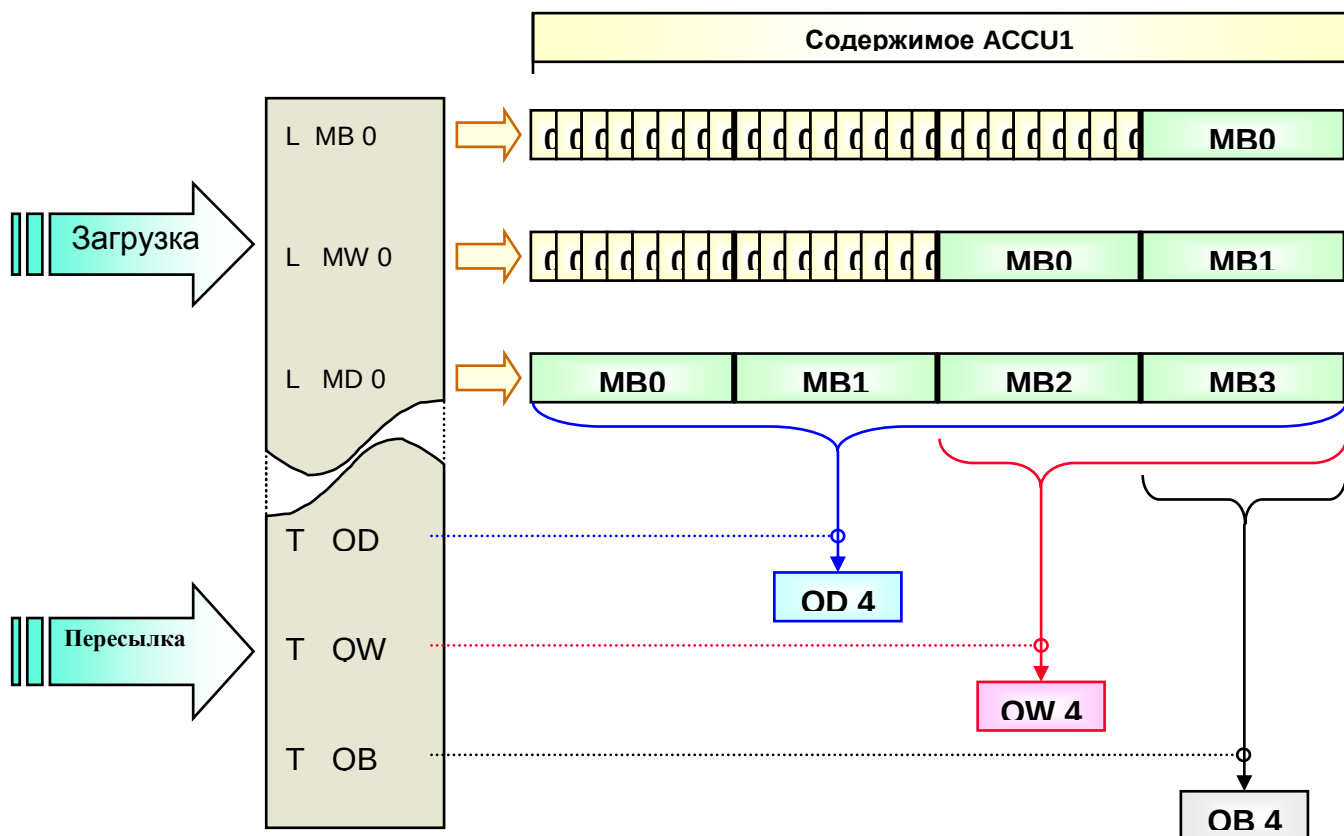


Рис. 4.2. Загрузка и выгрузка байтов, слов, двойных слов

При загрузке байта его содержимое считывается в аккумулятор 1 с выравниванием вправо. Неиспользуемые байты заполняются нулями.

При загрузке слова его содержимое считывается в аккумулятор 1 с выравниванием вправо. При этом (n+1)-ый байт слова располагается в аккумуляторе с выравниванием вправо, тогда как байт (n) слова располагается в аккумуляторе вплотную слева от байта (n+1). Неиспользуемые байты заполняются нулями.

При загрузке двойного слова его содержимое считывается в аккумулятор 1 с выравниванием вправо.

Ниже приведены примеры команд.

```
L IB n      //Загрузка данных из входного байта
L IW n      //Загрузка данных из входного слова
L ID n      //Загрузка данных из входного двойного слова
```

В некоторых типах CPU загрузка данных со входов разрешена, только если к соответствующим входным модулям имеется доступ.

```
L QB n      //Загрузка данных из выходного байта
L QW n      //Загрузка данных из выходного слова
L QD n      //Загрузка данных из выходного двойного слова
```

В некоторых типах CPU загрузка данных от выходов разрешена, только если к соответствующим выходным модулям имеется доступ.

```
L PIB n     //Загрузка данных из периферийного входного байта
L PIW n     //Загрузка данных из периферийного входного слова
L PID n     //Загрузка данных из периферийного входного двойного слова
```

При загрузке данных от области I/O входные модули адресуются как периферийные входы (PI). При этом доступны только существующие входные модули. Надо отметить, что при непосредственной загрузке данных от I/O модуля пересылаемое значение может отличаться от значения, загружаемого из области входов образа процесса с тем же адресом, так как эти значения одинаковы лишь в начале цикла сканирования программы (когда CPU обновляет отображение процесса). Непосредственная же загрузка данных от I/O модулей позволяет записать в аккумулятор "текущие" значения сигналов входов.

```
L MB n      //Загрузка данных из байта
L MW n      //Загрузка данных из слова
L MD n      //Загрузка данных из двойного слова памяти
```

Загрузка данных из области меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

Вы можете загружать константу или фиксированное значение непосредственно в аккумулятор. При этом для улучшения читаемости программы Вы можете выбирать для констант подходящее представление, использовать различные форматы. В главе 3 "SIMATIC S7-программа" Вы можете найти обзор разрешенных форматов для констант. Все константы, которые могут быть загружены в аккумулятор являются константами простых типов.

```
L B#16#F1   //Загрузка шестнадцатеричного числа - байта
L -1000      //Загрузка целого числа (INT)
L 5.0        //Загрузка действительного числа (REAL)
L S5T#2s     //Загрузка данных таймера формата S5
L C#250      //Загрузка числа формата BCD (значение счетчика)
L TOD#8:30:00 //Загрузка времени суток
```

В разделе "Типы данных" описывается назначение битов констант (показана битовая структура данных).

Указатели - это специальная форма констант, которая используется для вычисления позиции в памяти. Вы можете загружать следующие указатели в аккумулятор:

```
L P#1.0      //Загрузка внутризонного указателя (смещения)
L P#M2.1     //Загрузка межзонного указателя
L P#name     //Загрузка адреса локальной переменной
```

Вы не сможете загрузить указатель DB (DB pointer) или указатель ANY (ANY pointer) в аккумулятор, так как длина этих указателей превышает 32 бита.

4.3.3. Функции выгрузки данных из аккумулятора

Функция выгрузки состоит из оператора T (код операции выгрузки) и адреса в области памяти, по которому должны быть отправлены данные из аккумулятора accumulator 1:

```
T MW120      // Переносит содержимое аккумулятора по определенному адресу в память (абсолютная адресация)
T Setpoint   // Переносит содержимое аккумулятора в переменную (символьная адресация)
```

CPU выполняет функцию выгрузки независимо от результата логической операции (RLO) и битов состояния. Функция выгрузки не влияет на результат логической операции и не влияет на биты состояния.

Функция выгрузки пересылает содержимое аккумулятора 1 по одному байту, или по одному слову, или по одному двойному слову по заданному адресу. При этом содержимое аккумулятора 1 остается неизменным, что делает возможным многократную пересылку данных из аккумулятора 1 (см. Рис. 4.2).

Функция выгрузки возможна только с помощью аккумулятора 1. Если есть необходимость передать данные из другого аккумулятора, то Вы должны сначала передать эти данные в аккумулятор 1, используя функции аккумуляторов, и лишь затем переслать их с помощью функции выгрузки по требуемому адресу в память.

При выгрузке байта содержимое крайнего правого байта аккумулятора 1 пересылается в байт по указанному в выражении адресу.

При выгрузке слова содержимое крайнего правого слова аккумулятора 1 пересылается в слово по указанному в выражении адресу.

При выгрузке двойного слова его содержимое считывается из аккумулятора 1 пересылается в двойное слово по указанному в выражении адресу.

Примеры выгрузки данных приведены ниже.

```
T  IB n      // Выгрузка данных во входной байт
T  IW n      //Выгрузка данных во входное слово
T  ID n      //Выгрузка данных во входное двойное слово
```

В некоторых типах CPU выгрузка данных в входы разрешена, только если к соответствующим входным модулям имеется доступ. Пересылка во входы влияет только на биты в области отображения процесса, также как и операции установки и сброса этих битов.

Возможное применение для таких операций заключается в инициализации значений входов в целях отладки или для запуска с определенными начальными значениями: если Вы запускаете программу на выполнение с заданными значениями входов, то программа начинает выполняться именно с заданными Вами новыми значениями, а не со значениями, полученными от входных модулей.

```
T  QB n      //Выгрузка данных в выходной байт
T  QW n      //Выгрузка данных в выходное слово
T  QD n      //Выгрузка данных в выходное двойное слово
```

В некоторых типах CPU выгрузка данных в выходы разрешена, только если к соответствующим выходным модулям имеется доступ.

```
T  PQB n     //Выгрузка данных в периферийный выходной байт
T  PQW n     //Выгрузка данных в периферийное выходное слово
T  PQD n     //Выгрузка данных в периферийное выходное двойное слово
```

При выгрузке данных в область I/O выходные модули адресуются как периферийные выходы (PQ). При этом доступны только существующие выходные модули.

Надо отметить, что при непосредственной выгрузке данных в область I/O выходных модулей, связанных с таблицей выходов образа процесса, данные в этой таблице обновляются, ибо нет никакой разницы (в рассматриваемом контексте) между выходом (из области отображения процесса) и периферийным выходом с одинаковым адресом.

```
T  MB n      //Выгрузка данных в байт
T  MW n      //Выгрузка данных в слово
T  MD n      //Выгрузка данных в двойное слово
```

Выгрузка данных в область меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

4.3.4. Функции аккумуляторов

Функции аккумуляторов (Accumulator Functions) позволяют пересылать значения из одного аккумулятора в другой или перемещать байты внутри аккумулятора 1. На выполнение функций аккумуляторов не влияют ни результат логической операции RLO ни биты состояния. Также и результат выполнения этих функций не оказывает влияния ни на RLO, ни на биты состояния.

Рассмотрим прямую пересылку данных между аккумуляторами:

```
PUSH - сдвиг содержимого аккумулятора "вперед",
POP   - сдвиг содержимого аккумулятора "назад",
ТАК   - обмен содержимым между аккумуляторами 1 и 2,
ENT   - сдвиг содержимого аккумулятора "вперед" (без аккумулятора 1),
LEAVE - сдвиг содержимого аккумулятора "назад" (без аккумулятора 1).
```

Первые три функции PUSH, POP и TAK используются в CPU, имеющих только два аккумулятора (S7-300 CPU). Все пять функций используются в CPU, имеющих четыре аккумулятора (S7-400 CPU). Схематически выполнение перечисленных выше функций показано на рис. 4.3.

Функция **PUSH** вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 1, accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора 3 в аккумулятор 4, данных из аккумулятора 2 в аккумулятор 3 и копирование данных из аккумулятора 1 в аккумулятор 2 (содержимое аккумулятора 1 остается неизменным).

Таким образом, Вы можете использовать функцию PUSH для внесения одного и того же значения в несколько аккумуляторов.

Функция **POP** вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 4, accumulator 3, accumulator 2 и accumulator 1. При этом происходит перемещение данных из аккумулятора 4 в аккумулятор 3, данных из аккумулятора 3 в аккумулятор 2 и данных из аккумулятора 2 в аккумулятор 1 (содержимое аккумулятора 4 при этом остается неизменным).

Таким образом, Вы можете использовать функцию POP для переноса значений из аккумуляторов 4, 3 и 2 в аккумулятор 1, откуда эти значения могут быть пересланы по нужным адресам.

Содержимое аккумуляторов после выполнения операций:

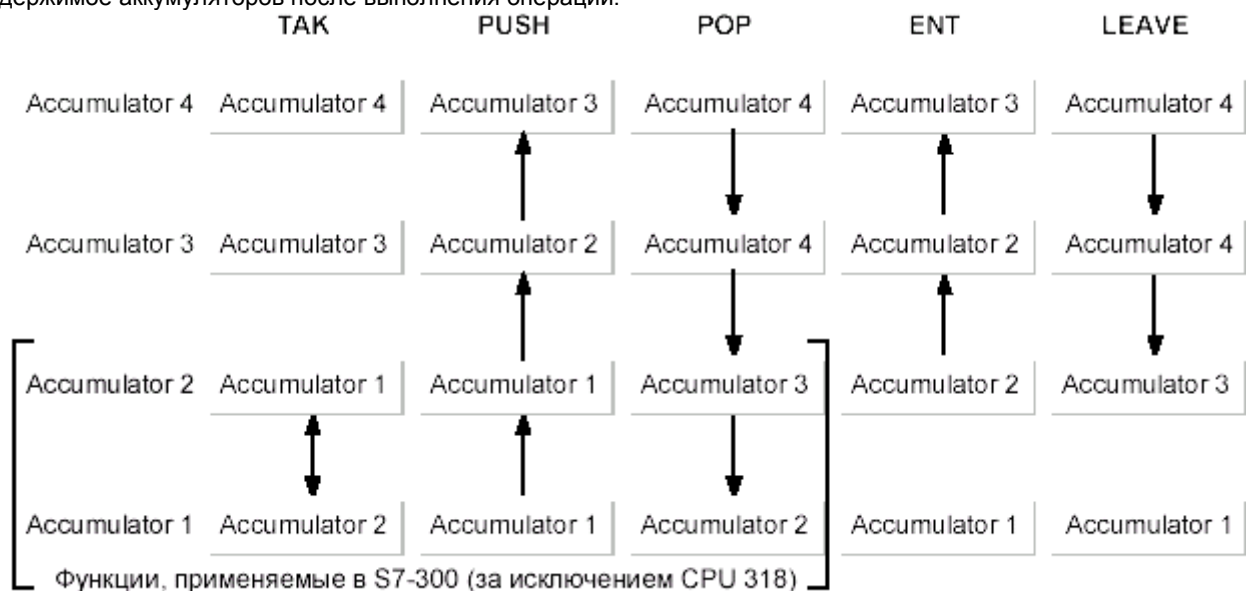


Рис. 4.3. Прямая пересылка данных между аккумуляторами в CPU S7-300 и S7-400

Функция **TAK** вызывает обмен содержимым между аккумуляторами 1 и 2. При этом содержимое аккумуляторов 3 и 4 остается неизменным.

Функция **ENT** вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора 3 в аккумулятор 4 и копирование данных из аккумулятора 2 в аккумулятор 3. При этом содержимое аккумуляторов accumulator 1 и accumulator 2 остается неизменным.

Если сразу за функцией ENT следует функция загрузки L, то они как бы вызывают сдвиг содержимого аккумуляторов вдоль всей цепочки аккумуляторов аналогично функции PUSH; при этом новое значение будет располагаться в аккумуляторе 1.

Функция **LEAVE** вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 4, accumulator 3 и accumulator 2. При этом происходит перемещение данных из аккумулятора 3 в аккумулятор 2 и копирование данных из аккумулятора 4 в аккумулятор 3. При этом содержимое аккумуляторов 4 и 1 остаётся неизменным.

Используя данную функцию Вы можете также смоделировать некоторые функциональные возможности других логических операций (например, в логических операциях с операндами формата слова).

Функция **SAW** меняет местами два байта в младшем слове в аккумуляторе 1. При этом байты старшего слова аккумулятора остаются неизменными.

Функция **CAD** меняет местами все байты в аккумуляторе 1. При этом самый старший байт становится самым младшим по номеру, а средние два байта меняются местами.

Схематически результат выполнения перечисленных выше функций показано на рис. 4.4.

4.3.5. Системные функции для пересылки данных

Для пересылки больших объёмов данных используются системные функции:

- SFC 20 (BLKMOV) - системная функция копирования области данных,
- SFC 21 (FILL) - системная функция вставки данных в область назначения,
- SFC 81 (UBLKMOV) - системная функция непрерывного копирования области данных.

Каждая из этих системных функций имеет два параметра типа ANY (см. табл. 4.1). Теоретически каждый из этих параметров может определять произвольный адрес, переменную или абсолютный адрес в памяти.

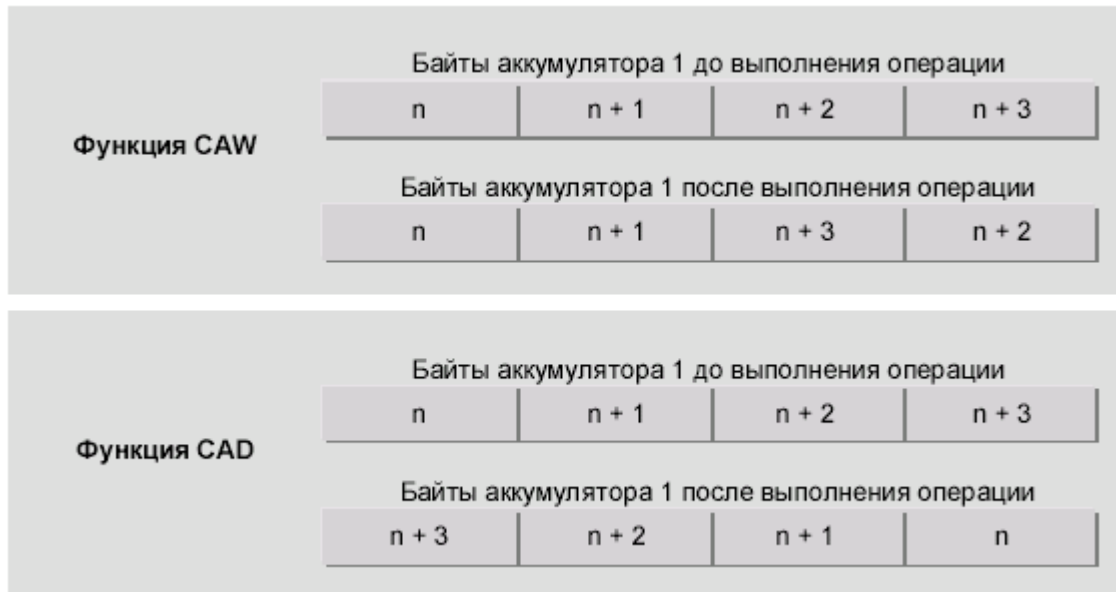


Рис. 4.4. Обмен байтов местами в аккумулятре accumulator 1

Таблица 4.1 Параметры для системных функций SFC 20, 21 и 81

SFC	Параметр	Назначение	Тип данных	Содержание, описание
SFC 20	SRCBLK	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DSTBLK	OUTPUT	ANY	Область-приемник, в которую копируются данные
SFC 21	BVAL	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	BLK	OUTPUT	ANY	Область-приемник, в которую копируются данные (включая повторное копирование)
SFC 81	SRCBLK	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DSTBLK	OUTPUT	ANY	Область-приемник, в которую копируются данные

Если Вы определяете переменную сложного типа, она должна быть определена целиком ("complete variable"); поэтому компоненты переменной (например, отдельные компоненты массива или структуры) не допускаются как параметры. Для абсолютной адресации Вы можете использовать указатель ANY; такие указатели рассматриваются в разделе "Указатели". Вы можете также использовать функции для копирования отдельных переменных типа STRING.

Если Вы используете временные локальные данные в виде фактического параметра блока типа ANY, редактор принимает такой параметр, как параметр со структурой указателя ANY. В таком случае Вы можете создать указатель ANY на временные локальные данные, которые могут быть изменены во время работы программы.

Системная функция SFC 20 позволяет выполнить копирование содержания области данных в направлении возрастания (инкрементного) их адресов (параметр SRCBLK) в область назначения (параметр DSTBLK). При этом могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),

- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 20 Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O), а также с помощью этой функции нельзя передавать информацию из/в системные блоки данных (SDB).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей. Вы можете также определить переменную или адрес области в блоке данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

При использовании функции SFC 20 BLKMOV область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: Переменная *Frame* (например, структурированная переменная пользовательского типа) в блоке данных "Rec_mailb" должна быть скопирована в переменную *Frame1* (такого же типа как и переменная *Frame*) в блоке данных "Buffer". Значение функции (в данном случае это код ошибки) должно быть введено в переменную *Copyerror* в блоке данных "Evaluation".

```
CALL BLKMOV
SRCBLK := Rec_mailb.Frame
RET_VAL := Evaluation.Copyerror
DSTBLK := Buffer.Frame1
```

Системная функция SFC 81 (UBLKMOV) позволяет копировать содержимое исходной области данных (параметр SRCBLK) в область назначения (параметр DSTBLK) в направлении возрастания (инкрементного) адресов данных. Операция копирования не может прерываться, поэтому в соответствующих условиях время ожидания обработки прерывания может возрасти. Объем копируемых данных может достигать 512 байт. Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 81 Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавать информацию из/в системные блоки данных (SDB) и из/в блоки данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 81 область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: из блока данных "Buffer" первый компонент массива *Data* должен быть скопирован в переменную *Frame* в блоке данных "Send_mailb". Значение функции (код ошибки) должно быть сохранено в переменной *Copyerror* в блоке данных "Evaluation".

```
CALL UBLKMOV
SRCBLK := Buffer.Data[1]
RET_VAL := Evaluation.Copyerror
DSTBLK := Send_mailb.Frame1
```

Системная функция SFC 21 (FILL) позволяет копировать содержимое исходной области данных (параметр BVAL) в область назначения (параметр BLK) в направлении возрастания адресов данных. Операция копирования продолжается до полного заполнения области назначения (при условии превышения размера области назначения над областью-источником возможно многократное копирование исходных данных). Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 21 Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавать информацию из/в системные блоки данных (SDB). Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 21 область-источник и область-приемник не должны перекрываться. Если область-источник больше области-приемника, то объем данных, переданный в область-приемник будет соответствовать размеру последнего; если же область-источник меньше области-приемника, то в область-приемник будут записываться (возможно многократно) копируемые данные до его полного заполнения (даже, если размеры рассматриваемых областей не кратны друг другу).

Пример: блок данных DB 13 содержит 128 байт. Необходимо скопировать во все эти байты содержимое байта памяти MB 80.

```
CALL SFC 21
BVAL := MB 80
RET_VAL := MW 32
BLK := P#DB13.DBX0.0 BYTE 128
```

Вы можете копировать отдельные переменные типа STRING с использованием системных функций SFC 20 и SFC 81.

Редактор STL-программ обрабатывает переменную типа STRING как массив байтов, так что SFC передает отдельные байты один к одному (включая два первые байта со спецификацией размера). Если, например, Вы пересылаете массив байтов в переменную типа STRING, то Вы должны задать правильную длину в первых двух байтах STRING-переменной ("length bytes") в соответствии с передаваемым массивом. Системные функции копируют STRING-переменную в другую STRING-переменную вполне корректно.

4.4. Функции таймеров

Функции таймеров используются для управления по времени, например, для обеспечения заданного времени ожидания (waiting) или мониторинга, для измерения отрезков времени или для генерации импульсов. Пользователю доступны следующие типы таймеров:

- таймер с управляемым импульсом (Pulse timer),
- таймер с расширенным импульсом (Extended pulse timer),
- таймер с задержкой включения (On-delay timer),
- таймер с задержкой включения с памятью (Retentive On-delay timer),
- таймер с задержкой выключения (Off-delay timer).

При запуске таймера Вы можете задавать динамические характеристики (dynamic response), длительность (duration), длительность задержки запуска/выключения таймера. Вы также можете выключить или включить функцию перезапуска ("retrigger") таймеров. Для опроса таймеров используются двоичные логические операции. Функции загрузки L используются для пересылки текущего значения времени таймера в двоичном или BCD-коде в аккумулятор 1.

4.4.1 Запуск таймера

Таймер запускается (т.е. начинает выполняться функция таймера), если перед выполнением CPU инструкции запуска таймера произошло изменение значения результата логической операции (RLO). При этом для таймера с задержкой выключения ("Off-delay timer") RLO должен изменить свое состояние со значения "1" на "0", а для всех остальных типов таймеров RLO должен изменить свое состояние со значения "0" на "1".

Вы можете запустить на выполнение любой из пяти возможных таймеров (см. рис. 4.5).

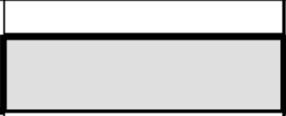
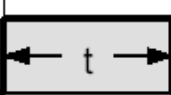
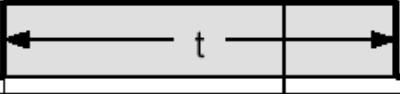
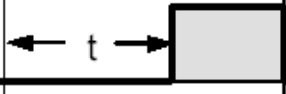
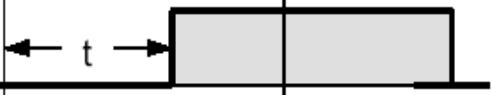
Таймер запускается как:	Ha STL	Ha SCL	Сигнал запуска		
Таймер с управляемым импульсом (импульсный) (Pulse timer)	SP	S_PULSE			
Таймер с расширенным импульсом (Extended pulse timer)	SE	S_PEXT			
Таймер с задержкой включения (On-delay timer)	SD	S_ODT			
Таймер с задержкой включения с памятью (Retentive on-delay timer)	SS	S_ODTS			
Таймер с задержкой выключения (Off-delay timer)	SF	S_OFFDT			

Рис. 4.5. Инструкции запуска для таймеров

4.4.2. Задание временных параметров таймера

При запуске таймера из аккумулятора 1 выбирается время запуска (running time) или длительность работы (duration). Как и когда в аккумулятор 1 поступают временные параметры таймера не имеет значения. Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор 1 непосредственно перед запуском функции таймера в виде константы (непосредственно заданная величина) или в виде переменной (например, слово в памяти, содержащее значение).

Примечание: аккумулятор 1 должен содержать корректное значение, даже если отсчет времени не начинается при выполнении инструкции запуска.

Примеры загрузки значений времени.

L S5TIME#10s //Длительность 10 с

L S5T#1m10ms //Длительность 1 мин и 10 мс

В языке STL параметры "время запуска" (running time) и "длительность работы (импульса)" (duration) могут быть определены в часах, минутах, секундах и миллисекундах. Диапазон задания временных параметров простирается от S5TIME#10ms до S5TIME#2h46m30s (что соответствует диапазону времени, равному 9990 с). При этом Вы можете использовать как формат S5TIME#, так и формат S5T# для определения временных параметров с помощью константы.

Пример определение длительности работы (импульса) таймера с помощью переменной:

L S5T#10m //Длительность (duration) = 10 мин

T MW20 //Сохранить "длительность работы"

...

L MW20 //Загрузить "длительность работы"

Внутренняя структура временного параметра "длительность работы (импульса)" состоит из "значения времени" и "временной базы". Длительность работы или импульса таймера равна произведению этих величин: "длительность работы (импульса)" = "значение времени" x "временная база".

Длительность работы (импульса) таймера - это период времени, в течение которого таймер находится в активном состоянии. Значение времени представляет собой число отрезков времени в сумме дающих полный период времени, в течение которого таймер должен находиться в активном состоянии. Величина такого отрезка времени равна значению "временной базы" ("time base") и является величиной шага по времени, которая используется операционной системой CPU для "декрементирования" таймера (рис. 4.6).



Рис. 4.6. Назначение битов параметра "длительность работы"

Вы можете непосредственно задать параметр "длительность работы" в машинном слове. Наименьшее возможное значение для "временной базы" обеспечивает более точное исчисление промежутков времени с помощью таймера. Например, если необходимо задать для таймера отрезок времени, равный 1 с, то можно при этом использовать одно из трех значений для "временной базы" ("time base") и, соответственно, для каждого из этих значений Вы получите свое значение длительности работы функции таймера (фактически, значение функции таймера):

"Временная база" = 1 с; "Длительность работы" = 2001_{hex}

"Временная база" = 100 мс; "Длительность работы" = 1010_{hex}

"Временная база" = 10 мс; "Длительность работы" = 0100_{hex}

Последний из трех вариантов является предпочтительным для данного случая. При запуске таймера CPU использует значение времени как период, в течение которого таймер находится в активном состоянии. Операционная система обновляет таймеры через фиксированные интервалы времени независимо от процесса сканирования программы пользователя. То есть, CPU производит ступенчатое уменьшение значения функции таймера в соответствии с заданным значением для "временной базы".

Когда значение функции таймера достигает уровня "0", это означает, что отсчитываемое таймером время истекло. При этом CPU изменяет состояние (статус) таймера (состояние сигнала в "0" или "1", в зависимости от выбранного типа таймера), при этом функция таймера перестает быть активной до следующего запуска таймера.

Если для функции таймера было задано значение времени, равное нулю (0), то таймер остается активным, пока при обработке функции таймера CPU не обнаружит, что время заданное для таймера "истекло".

Таймеры обновляются асинхронно по отношению к процессу сканирования программы пользователя. Следовательно, возможно, что состояние таймера в начале цикла сканирования отличается от его состояния в конце цикла.

Если Вы используете функцию таймера только в одном месте программы и придерживаетесь нижеследующих рекомендаций, то не возникнет ошибок из-за асинхронного обновления таймера в программе.

4.4.3. Сброс таймера (Resetting a timer)

Инструкция **R T n** вызывает сброс таймера, где **n** – целое число (номер таймера).

Таймер сбрасывается, при RLO равном "1" перед вышеуказанной инструкцией. После сброса таймер пока RLO равен "1", проверки таймера на состояние "1" возвращают результат "0"; проверки таймера на состояние "0" возвращают результат проверки "1".

Сброс таймера устанавливает для значения времени и для "временной базы" нулевые значения (0).

Примечание: Сброс функции таймера не сбрасывает внутренний меркер фронта для запуска. Для повторного запуска CPU должен обработать инструкцию запуска таймера при RLO, равном "0", и только после этого при появлении фронта сигнала таймер сможет стартовать.

4.4.4. Разблокировка таймера (Enabling a timer)

Инструкция **FR T n** позволяет «перезапустить» таймера.

Формат: **FR <timer>**

При переходе RLO из "0" в "1", **FR <timer>** обнуляет флаг выделения фронта, используемый для запуска адресуемого таймера. Изменение RLO с 0 на 1 перед инструкцией деблокировки (FR) запускает любую последующую команду таймера, если перед ней RLO = 1.

Деблокировка не обязательна для нормального запуска таймера с помощью соответствующих инструкций. Она может быть необходима только для перезапуска работающего таймера.

Деблокировка сбрасывает биты **IFC** и **OR** слова статуса.

Пример:

A I 2.0 //фронт для FR

FR T1 //Деблокировка таймера T1

```

A I 2.1      // RLO = 1 и фронт перед FR T1 разрешают выполнение команды SP T1
L S5T#10s    //Задание времени 10 секунд в ACCU 1
SP T1        //Запуск T1 как импульсного
A I 2.2      // RLO = 1 разрешает выполнение команды сброса T1
R T1         //Сброс T1
A T1         //Опрос таймера T1
= Q 4.0
L T1         //Загрузка текущего значения таймера T1 в двоичном коде
T MW10
...

```

Команды для таймера в примере выполняются, если инструкция FR обрабатывается при положительном (возрастающем) фронте сигнала. При этом внутренний меркер фронта сбрасывается для обеспечения возможности запуска таймера. Если после этого результат логической операции RLO становится равным "1" перед появлением вышеуказанной инструкции, то таймер запускается, даже если при этом не определяется фронт сигнала.

Сброс таймера вызывает изменение величин времени и "временной базы" на нулевые значения (0).

4.4.5. Проверка (опрос) таймера

Перечисленные ниже команды показывают все варианты опроса таймера:

```

A T n - опрос на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И);
O T n - опрос на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ);
X T n - опрос на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR
(Исключающее ИЛИ);
AN T n - опрос на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И);
ON T n - опрос на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ);
XN T n - опрос на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR
(Исключающее ИЛИ).

```

Вы можете опрашивать таймер, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. В зависимости от типа таймера проверка (опрос) сигнала на состояние "1" вызывает разное поведение сигнала (импульса) таймера во времени (см. далее описание динамических характеристик).

Как и в случае с опросом входа (input) проверка (опрос) сигнала на состояние "0" возвращает результат проверки таймера с точностью до наоборот по сравнению с опросом на состояние "1" - результат проверки инвертируется.

Инструкция

```

L Tn - загружает двоичное значение времени таймера,
LC Tn - загружает значение времени таймера в двоично-десятичном формате (BCD).

```

Функции загрузки L и LC считывают определенное значение времени таймера и пересылают его в аккумулятор 1. При этом инструкция L загружает значение времени в двоичном, а инструкция LC загружает значение времени таймера в двоично-десятичном формате. Значение времени таймера, пересылаемое в аккумулятор 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос таймера (если функция таймера находится в активном состоянии, то значение времени, которое загружается в аккумулятор, будет одним из значений, полученным CPU при уменьшении заданного исходного значения).

Значение времени, определенное с помощью инструкции таймера, имеет двоичный формат, и может быть переслано в аккумулятор accumulator 1 в этом же (двоичном) формате. В этом случае "временная база" ("time base") будет потеряна и на ее месте в аккумуляторе accumulator 1 будут записаны нули "0". Следовательно, в аккумуляторе будет значение, которое соответствует положительному целому (INT) числу и которое может быть обработано в дальнейшем с помощью, например, функций сравнения.

Примечание: это значение таймера не "заданная длительность работы", а оставшееся время.

Пример:

```

L T 15      //загрузка текущего значения времени таймера вместе с базой
T MW 34     //сохранение текущего значения времени

```

Вы можете также использовать инструкцию для т.н. "кодированной" пересылки в аккумулятор двоичного значения времени таймера. В этом случае и значение времени и "временная база" будут иметь формат BCD. При этом также как и в предыдущем случае в содержимом аккумулятора 1 в левое машинное слово (в старшее слово) будут записаны нули "0".

Пример:

LC T 16; //загрузка текущего значения времени таймера в BCD-формате
T MW 122; //сохранение текущего значения времени ("time value")

4.4.6. Последовательность инструкций при использовании функций таймера

При программировании таймера Вам нет необходимости использовать все возможные выражения для активации функций таймеров. Вы должны использовать только те из функций, которые Вам необходимо выполнить.

Обычно используется таймер с заданной длительностью импульса и двоичный опрос состояния таймера. Чтобы выполнить функцию таймера в соответствии с описанием в предыдущих разделах необходимо соблюдать определенный порядок при программировании соответствующих операторов. В таблице 4.2 показан «оптимальный» порядок для всех операторов при программировании функций таймера.

Таблица 4.2 Последовательность операторов для таймера

Функция таймера	: Примеры:
Разблокировка таймера (Enable timer)	A I 16.5 FR T 5
Запуск таймера (Start timer)	A I 17.5 L S5T#1s SP T 5
Сброс таймера (Reset timer)	A I 18.0 R T 5
Опрос численного значения таймера	L T 5 T MW20 LC T 5 T MW22
Двоичный опрос таймера	A T 5 = Q 2.0

При использовании операторов выбирайте те, которые необходимы Вам, и просто пропускайте ненужные.

Если таймер запускается и сбрасывается "одновременно" (как в показанной последовательности операторов), то функция таймера сначала будет запущена на выполнение, а следующее выражение немедленно сбросит таймер. Следовательно, если после этого таймер будет проверен (опрошен), то факт запуска таймера останется незамеченным.

4.4.7. IEC-функции таймеров (IEC Timer Functions)

IEC-функции таймеров (IEC Timer Functions) встроены в операционную систему CPU как системные функциональные блоки SFB.

В соответствующим образом оснащенных CPU могут быть доступны следующие функции таймеров:

- SFB 3 (TP) - генератор импульсов,
- SFB 4 (TON) - генерация импульса с задержкой включения,
- SFB 5 (TOF) - генерация импульса с задержкой выключения.

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать эти SFB как локальные экземпляры в функциональном блоке.

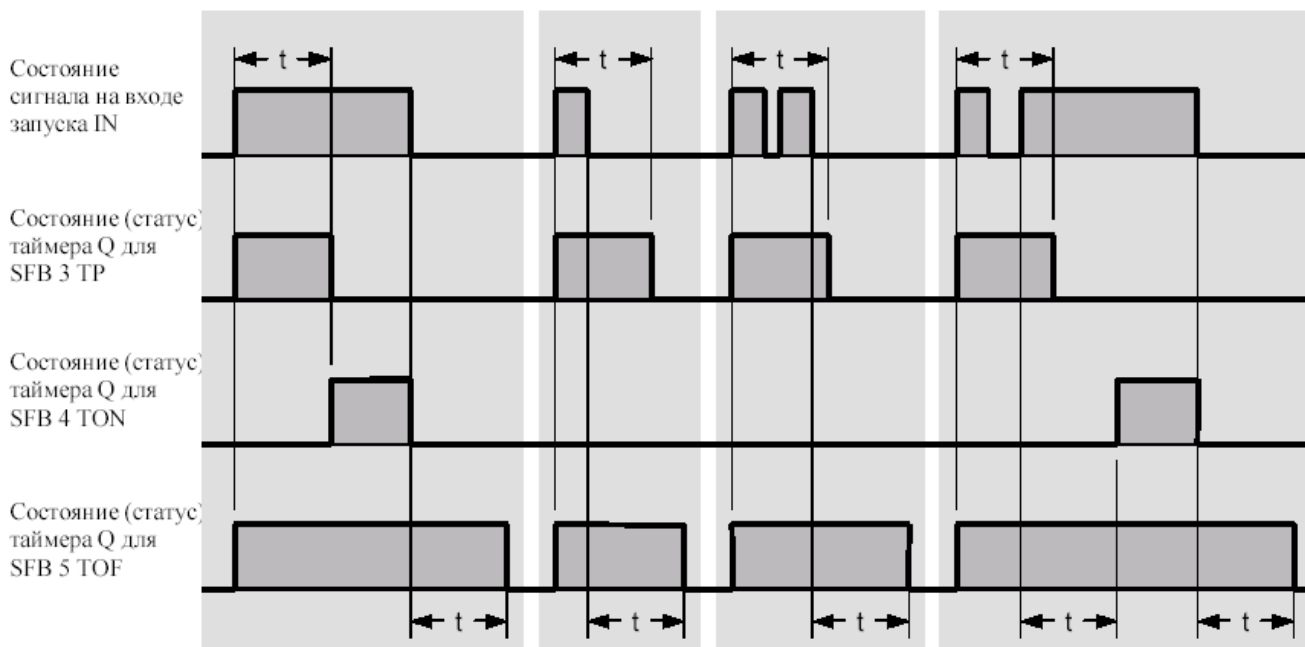


Рис. 4.7. Динамические характеристики IEC-функций таймеров

4.5. Функции счетчиков (Counter Functions)

Функции счетчиков позволяют решать задачи счета непосредственно в CPU. Счетчики позволяют выполнять прямой и обратный счет и используют при этом "трехдекадный" формат значения счетчика и диапазон значений от 000 до 999.

Скорость счета счетчиков зависит от времени сканирования Вашей программы! Для обеспечения процесса счета CPU должен обнаруживать на входе счетчика изменения входного сигнала, иначе говоря, входной импульс (или межуимпульсная пауза) на входе должна присутствовать по крайней мере один цикл сканирования программы. Чем продолжительнее цикл сканирования программы, тем медленнее скорость счета счетчика.

Примечание: в S7-300 CPU со встроенными функциями (CPU 3xxIFM) имеются встроенные функции счетчика, которые обеспечивают счет с использованием специального входа счетчика с частотой следования импульсов до 10 кГц. Значения счетчиков, описанных в данной главе, сохраняются в системной памяти CPU. Вы можете задавать для счетчика начальное значение. Вы также можете сбрасывать счетчик, включать режим прямого или обратного счета счетчика. Существует возможность определения состояния счетчика (содержит ли счетчик нулевое или ненулевое значение). Функции загрузки используются для пересылки текущего значения счетчика в двоичном или BCD-коде в аккумулятор 1.

4.5.1. Установка и сброс счетчиков

S Cn - команда записи в счетчик значения из аккумулятора 1. Она выполняется, если RLO переходит от "0" к "1" перед операцией S установки значения счетчика. Т.е. для задания значения счетчика всегда требуется положительный фронт.

Установить счетчик - это значит загрузить в счетчик начальное значение. Начальное значение, которое загружается в счетчик, должно находиться в аккумуляторе 1. Оно должно быть в диапазоне от 0 до 999.

Функция установки счетчика ("set counter") использует значение в аккумуляторе 1 как "значение счетчика". Как и когда это значение появляется в аккумуляторе 1 не имеет значения. Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор 1 непосредственно перед запуском функции счетчика или в виде константы (т.е., в виде непосредственно заданной величины), или в виде переменной (например, через слово в памяти, содержащее значение счетчика).

Примечание: аккумулятор 1 должен содержать корректное значение, даже если счетчик не устанавливается во время выполнения инструкции.

Примеры: определение (спецификация) счетчика с помощью константы:

L C#100 //Значение для счетчика – 100

L W#16#0100 //Значение для счетчика - 100

Значения счетчика лежат в диапазоне 000 - 999. При этом используются три декады. Значение счетчика может быть только в формате BCD. При этом счетчики не могут работать с отрицательными числами.

Для задания константы Вы можете использовать C# или W#16# (и только с десятичными числами).

R Cn - сброс счетчика. Счетчик сбрасывается, если RLO имеет значение "1" перед тем, как в программе встретится операция R сброса счетчика. Пока RLO равен "1" перед командой сброса, проверки счетчика на состояние "1" возвращают результат "0"; проверки счетчика на состояние "0" возвращают результат "1". Сброс устанавливает для значения счетчика нулевое значение (0).

Примечание: сброс счетчика не сбрасывает внутренний меркер фронта для установки счетчика, для включения режима прямого счета и для включения режима обратного счета. Для выполнения команд загрузки счетчика CPU должен обработать инструкцию установки или запуска на счет при RLO, равном "0", и только после этого при появлении фронта сигнала в соответствующем меркере фронта счетчик может быть установлен или изменен. Также для повторной активации функций счетчика Вы можете использовать операцию разблокировки (деблокировки) счетчика (см. ниже).

4.5.2. Счет (Counting)

Инструкция **CU Cn** вызывает процесс прямого счета. Счетчик выполняет прямой счет ("инкрементируется"), если инструкция CU обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CU, увеличивает значение счетчика ("count value") на единицу, пока не будет достигнут верхний предел, равный 999. После этого положительный фронт сигнала RLO перед вводом CU никак не будет влиять на состояние счетчика.

Инструкция **CD Cn** вызывает процесс обратного счета. Счетчик выполняет обратный счет ("декрементируется"), если инструкция CD обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CD, уменьшает значение счетчика ("count value") на единицу, пока не будет достигнут нижний предел, равный 0. После этого положительный фронт сигнала RLO перед вводом CD никак не будет влиять на состояние счетчика.

Значения счетчика отрицательными быть не могут.

4.5.3. Проверка счетчика (Checking a Counter)

Проверка (опрос) состояния счетчика выполняется следующими командами:

A Cn - проверка счетчика на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И),

O Cn - проверка счетчика на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ),

X Cn - проверка счетчика на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ),

AN Cn - проверка счетчика на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И),

ON Cn - проверка счетчика на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ),

XN Cn - проверка счетчика на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ).

Вы можете опрашивать счетчик, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. Проверка (опрос) счетчика на состояние "1" вызывает результат "1", если значение счетчика больше 0 и результат, равный "0", если значение счетчика равно 0.

Инструкция **L Cn** непосредственно загружает двоичное значение счетчика в аккумулятор 1 в форме двоичного числа. Значение счетчика, пересылаемое в аккумулятор 1, представляет собой текущее значение счетчика. Теперь значение в аккумуляторе 1 соответствует положительному числу целого типа (INT) и может использоваться для дальнейшей обработки, например, с помощью арифметических функций.

Пример:

L C99 //загрузка текущего значения счетчика

T MW 76 //сохранение текущего значения счетчика

Загрузка значения счетчика в BCD-формате ("coded loading") выполняется инструкцией **LC Cn**, которая загружает двоично-десятичное ("кодированное") значение счетчика в аккумулятор 1. Значение счетчика, пересылаемое в аккумулятор 1, представляет собой текущее значение счетчика. Содержимое аккумулятора 1 доступно для дальнейшего использования в виде числа в BCD-формате, выровненного вправо. Оно имеет такую же структуру как и заданное значение счетчика.

Пример:

LD C 99; //загрузка текущего значения счетчика в BCD-формате

T MW 50; //сохранение текущего значения счетчика

4.5.4. Деблокировка счетчика

Инструкция **FR Cn** позволяет выполнить «перезапуск» счетчика. Изменение с "0" на "1", результата логической операции перед командой деблокировки **FR <counter>** активирует счетчик, сбрасывая биты выделения фронта для установки счетчика и счета на увеличение и уменьшение. На самом деле фронт перед командой **FR** действует как фронт перед любой последующей командой счётчика, если перед ней **RLO=1**. Если перед командой счётчика (не **FR**) **RLO=0**, то **FR** никак не влияет на счётчик.

Деблокировка счетчика не требуется ни для установки счетчика, ни для нормального счета.

Команда FR сбрасывает биты слова статуса /FC и OR и не влияет на остальные.

Примечание: инструкция деблокировки функций счетчика воздействует на функции установки, запуска на счет и сброса счетчика одновременно!

Ниже представлен пример, показывающий принципы работы инструкции деблокировки:

```
A "Enable"
FR "Counter" // при наличии фронта влияет на все последующие инструкции для "Counter"
A "Count up"
CU "Counter" // зависит от FR
A "Count down"
CD "Counter" // зависит от FR
A "Set"
L C#020
S "Counter" // зависит от FR
A "Reset"
R "Counter"
A "Counter"
= "Counter status"
```

4.5.6 IEC-функции счетчиков

IEC-функции счетчиков (IEC Counter Functions) встроены в операционную систему CPU как системные функциональные блоки SFB. В соответствующим образом оснащенных CPU могут быть доступны следующие функции счетчиков:

- SFB 0 (CTU) - функция прямого счета,
- SFB 1 (CTD) - функция обратного счета,
- SFB 2 (CTUD) - функция прямого и обратного счета.

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать их как локальные экземпляры в функциональном блоке.

Вы можете найти описание интерфейса функций для программирования в автономном режиме (offline) в "стандартной библиотеке" Standard Library в разделе System Function Blocks.

.

4.6. Функции для работы с числами

Функции для работы с числами используются для обработки численных значений преимущественно типов INT, DINT и REAL, что значительно расширяет функциональные возможности PLC. Здесь будут рассмотрены функции для обработки чисел, используемые в STL-программах.

Функции сравнения формируют двоичный результат сравнения двух величин. Функции сравнения могут обрабатывать данные типов INT, DINT и REAL.

Арифметические функции позволяют выполнять в программе вычисления. Все базовые арифметические операции могут быть выполнены с данными типов INT, DINT и REAL.

Математические функции расширяют собой вычислительные возможности в программе. Математические функции добавляют к базовым арифметическим функциям, в частности, тригонометрические функции.

Функции преобразования позволяют до и после выполнения вычислений приводить численные значения к требуемому типу данных.

Функции сдвига позволяют побитно смещать содержимое аккумулятора влево или вправо. Функции сдвига позволяют также проверить бит, смещенный последним.

Логические функции с данными формата Word используются для маскирования численных значений, для проверки отдельных битов и задания для них значений "0" или "1".

Функции для обработки чисел работают, главным образом, со значениями в блоках данных. Это могут быть блоки глобальных данных или экземплярные блоки данных, если используются статические локальные данные.

За исключением аккумуляторов для хранения временных результатов очень удобны временные локальные данные.

4.6.1. Функции сравнения

Функции сравнения обеспечивают выполнение операции сравнения двух численных значений, одно из которых находится в аккумуляторе 1, а второе находится в аккумуляторе 2. После выполнения операции сравнения функции сравнения устанавливают RLO (результат логической операции) и биты состояния CC0 и CC1. Этот результат может в дальнейшем быть использован в двоичных логических операциях, в операциях с памятью или в операторах перехода. В таблице 4.3 Вы найдете обзор доступных пользователю функций сравнения.

Таблица 4.3. Общее представление функций сравнения

Операция сравнения	Типы данных		
	INT	DINT	REAL
Равно	==I	==D	==R
Не равно	<>I	<>D	<>R
Больше чем	>I	>D	>R
Больше чем или равно	>=I	>=D	>=R
Меньше чем	<I	<D	<R
Меньше чем или равно	<=I	<=D	<=R

В разделе "Биты состояния" будет показано, как функции сравнения устанавливают биты состояния CC0 и CC1. В этом разделе будут рассмотрены функции для обработки чисел, используемые в языке программирования STL.

Вы можете программировать функции сравнения в соответствии со следующей общей схемой:

```
загрузка адреса Address1;
загрузка адреса Address2;
функция сравнения;
присвоение результата Result.
```

При выполнении первой операции загрузки из первого адреса число помещается в аккумулятор 1. При загрузке из второго адреса содержимое аккумулятора 1 перемещается в аккумулятор 2 (см. раздел "Операции загрузки"). Теперь с содержимым двух аккумуляторов можно выполнить операции с помощью функций сравнения.

Функции сравнения возвращают двоичный результат (тип BOOL), который заносится в RLO и может быть назначен двоичному адресу или может быть использован в какой-либо другой двоичной проверке (опросе).

Функции сравнения не изменяют содержимого аккумуляторов. Они всегда выполняются вне всякой связи с какими-либо условиями.

Примеры функций сравнения:

```
L Arg1
L Arg2
==I
= M10.1
```

```
<I
= M10.2
>I
= M10.3
```

В случае, когда данные имеют тип INT, CPU сравнивает только расположенные справа слова (младшие слова) в аккумуляторах; содержимое слов, расположенных слева (старшие слова), в расчет не берется. В случае, когда данные имеют тип REAL, выполняется проверка для определения корректности (в соответствии с типом данных) содержимого аккумуляторов. Если результат этой проверки отрицателен, то CPU сбрасывает RLO в состояние "0", а биты состояния CC0, CC1, OV и OS устанавливает в состояние "1".

Инструкция для сравнения чисел с целью определения их равенства интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить равенство двух числовых величин. После выполнения инструкции для RLO устанавливается значение "1" в случаях, если содержимое аккумулятора 2 равно содержимому аккумулятора 1 для данных типа INT и DINT; если содержимое аккумулятора 2 равно содержимому аккумулятора 1 для данных типа REAL, при условии, что оба аккумулятора содержат корректные действительные (REAL) числа; в остальных случаях RLO сбрасывается (= "0").

Инструкция для сравнения чисел с целью определения их неравенства интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить неравенство двух числовых величин. После выполнения инструкции для RLO устанавливается значение "1" в следующих случаях:

- если содержимое младшего слова аккумулятора 2 не равно содержимому младшего слова аккумулятора 1 для данных типа INT,
 - если содержимое аккумулятора 2 не равно содержимому аккумулятора 1 для данных типа DINT,
 - если содержимое аккумулятора 2 не равно содержимому аккумулятора 1 для данных типа REAL при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.
- Если два числа формата REAL не равны, но одно из них или оба некорректны, то условие неравенства не выполняется (RLO = "0").

Инструкции для сравнения с целью определения выполнения условия "больше", "больше или равно", "меньше" и "меньше или равно" интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить, выполняется ли данное отношение для чисел, находящихся в аккумуляторах 2 и 1. После выполнения инструкции сравнения RLO устанавливается в "1" по аналогии с описанным выше отношением «не равно».

Функции сравнения возвращают двоичный результат логической операции RLO и, следовательно, могут быть использованы в сочетании с другими двоичными функциями. Функции сравнения устанавливают бит слова состояния /FC, например, в логических операциях функция сравнения всегда является первичным опросом ("first check").

Пример: функция сравнения в начале логической операции.

```
L MW 120
L 512
>I
A Input1
= Output1
```

В примере выход *Output1* устанавливается, если выполняется условие сравнения функции сравнения, а вход *Input1* имеет состояние "1".

Пример: функция сравнения внутри логической операции должна быть заключена в скобки, так как функция сравнения начинает новый логический шаг.

```
O Input2
O(
L MW 122
L 200
<=I
)
O Input3
= Output2
```

В примере выход *Output2* устанавливается, если вход *Input2* или вход *Input3* имеет состояние "1", или если выполняется условие сравнения.

Пример: многократное использование функции сравнения. Так как функции сравнения не изменяют содержимое аккумуляторов, возможно многократное использование этих функций при программировании на STL.

```
L MW 124
L 1200
>I
```

JC GRE

==I

JC EQU

В примере использованы две функции сравнения для одного и того же содержимого аккумуляторов. Первая операция сравнения генерирует RLO = "1", если MW 124 больше чем 1200, поэтому выполняется переход к метке GRE. Без перезагрузки аккумуляторов вторая функция сравнения выполняет проверку условия равенства и генерирует новое значение RLO.

Функция сравнения устанавливает биты состояния, исходя из соотношения сравниваемых величин, т.е. независимо от типа проверяемого условия в операции сравнения. Вы можете использовать это, делая проверку битов состояния в соответствующих функциях перехода.

Вышеприведенный пример может быть также запрограммирован следующим образом:

L MW 124

L 1200

>I

JP GRE

JZ EQU

В этом примере результат выполнения функции сравнения проверяется с помощью битов состояния CC0 и CC1. Собственно условие сравнения ("больше чем") в данном случае не влияет на установку битов состояния; могут быть использованы и другие условия сравнения, например, "меньше чем". Оператор JP проверяет не является ли первая из сравниваемых величин больше, чем вторая. Оператор JZ проверяет, не являются ли обе сравниваемые величины равными.

4.6.2. Арифметические функции

Арифметические функции обеспечивают выполнение базовых арифметических операций с двумя числовыми значениями, одно из которых находится в аккумуляторе 1, а второе находится в аккумуляторе 2. Результат арифметической операции помещается в аккумулятор 1. Биты состояния CC0, CC1, OV и OS обеспечивают информацию, касающуюся выполнения вычислений и результата (см. раздел "Биты состояния"). В таблице 4.4 Вы найдете обзор доступных пользователю арифметических функций.

Таблица 4.4. Обзор арифметических функций

Арифметическая функция	Типы данных		
	INT	DINT	REAL
Сложение (Addition)	+I	+D	+R
Вычитание (Subtraction)	-I	-D	-R
Умножение (Multiplication)	*I	*D	*R
Деление (Division with quotient as result)	/I	/D	/R
Остаток от деления (Division with remainder as result)	-	MOD	-

В дополнение к базовым арифметическим операциям с участием числа, находящегося в аккумуляторе accumulator 2, Вы также можете прибавлять константы непосредственно к содержимому аккумулятора accumulator 1 или изменять его содержимое на некоторую фиксированную величину.

Вы можете программировать арифметические функции в соответствии со следующей общей схемой:

загрузка адреса Address1;

загрузка адреса Address2;

арифметическая функция;

передача результата Result;

При выполнении первой операции загрузки Load из первого адреса число помещается в аккумулятор accumulator 1. При загрузке из второго адреса содержимое аккумулятора 1 перемещается в аккумулятор 2 (см. раздел "Операции загрузки"). Теперь с содержимым двух аккумуляторов можно выполнить операции с помощью арифметических функций. Результат операции сохраняется в аккумуляторе accumulator 1.

Арифметические функции выполняются в соответствии с заданными параметрами. Они всегда выполняются вне всякой связи с какими-либо условиями.

В случае, когда данные имеют тип INT, арифметические операции выполняются только для расположенных справа слов (младшие слова) в аккумуляторах; слова, расположенные слева (старшие), игнорируются. В случае, когда данные имеют тип REAL, выполняется проверка для определения корректности (в соответствии с типом данных) содержимого аккумуляторов.

В S7-300 CPU (за исключением CPU 318) выполнение арифметических функций не изменяет содержимое аккумулятора accumulator 2; в S7-400 CPU и CPU 318 содержимое аккумулятора 2 после выполнения операции переписывается содержимым аккумулятора 3. При этом содержимое аккумулятора 4 смещается (заменяет собой прежнее значение) в аккумулятор 3 (см. рис. 4.8).

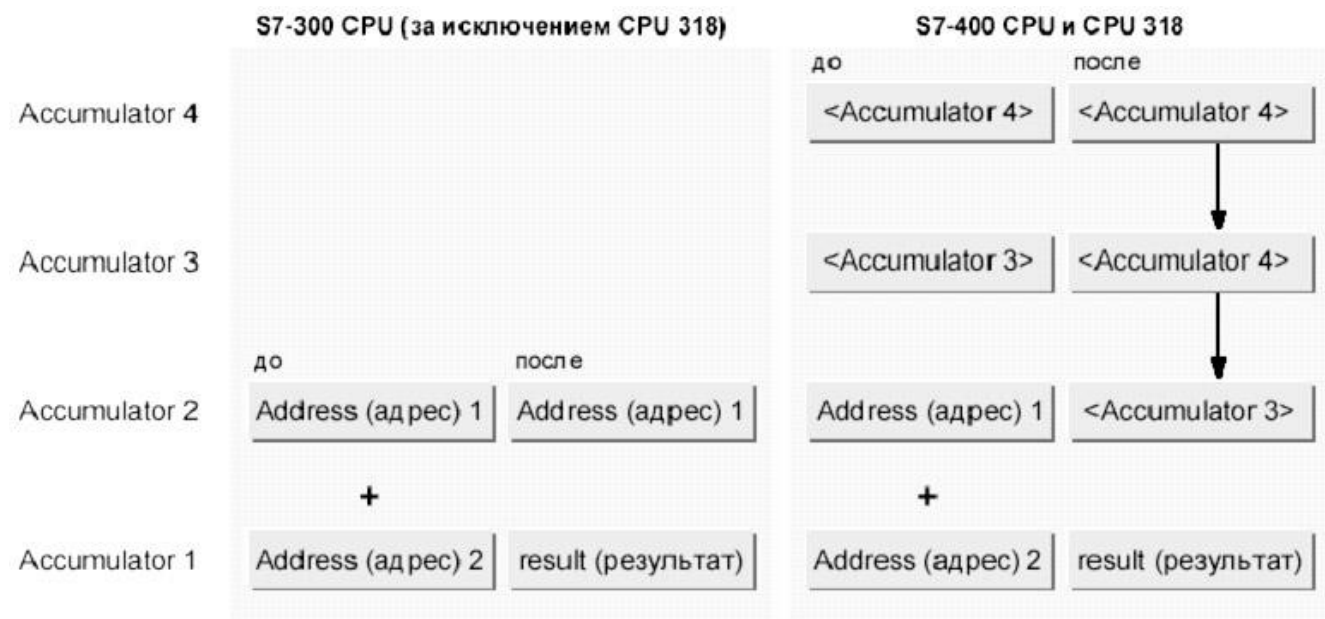


Рис. 4.8. Содержимое аккумуляторов при выполнении арифметических функций

Инструкции для арифметических функций с данными типа INT интерпретируют содержимое младших слов аккумуляторов 1 и 2 как числа целого типа (INT). Инструкция призвана выполнить арифметическую операцию с этими двумя числовыми величинами и сохранить результат в аккумуляторе 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков результат: отрицателен, равен нулю или положителен. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона. Биты слова состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Для операции перемножения результат, находящийся в аккумуляторе 1, имеет формат числа DINT.

После выполнения инструкции деления младшее слово аккумулятора 1 содержит частное от деления. Частное является целым результатом операции деления. Частное от деления равно нулю в двух случаях: если делимое равно нулю, в то время как делитель не равен нулю, или если делимое меньше чем делитель. Частное от деления отрицательно, если делитель меньше нуля. После выполнения инструкции старшее слово аккумулятора 1 содержит остаток от деления (не путать остаток от деления с дробной частью результата операции деления!). Остаток от деления отрицателен, если делимое меньше нуля.

В случае деления на ноль частное от деления и остаток возвращаются с нулевыми значениями, а биты слова состояния CC0, CC1, OV и OS устанавливаются в состояние "1".

При выполнении арифметических операций с данными типов DINT и REAL аккумуляторы используются как 32-хбитные регистры.

Инструкция MOD для нахождения остатка от деления интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить деление числа, находящегося в аккумуляторе 2 (делимое), на число, находящееся в аккумуляторе 1 (делитель), и сохранить остаток от деления в аккумуляторе 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков остаток от деления: отрицателен, равен нулю или положителен.

В случае деления на ноль остаток от деления возвращается с нулевым значением, а биты состояния CC0, CC1, OV и OS устанавливаются в "1".

4.6.3. Добавление констант к содержимому аккумулятора 1

Вы можете запрограммировать добавление константы в соответствии со следующей общей схемой:

загрузка адреса Address;

сложение с константой Const;

передача результата Result.

Сами команды имеют вид:

- + B#16#bb - добавление байтовой константы (byte),
- + ±w - добавление константы формата слово (word),
- + L#±d - добавление константы формата двойное слово (duobleword).

Операция добавления константы более предпочтительна для вычислений с адресуемыми данными, потому что по сравнению с арифметическими функциями она не влияет ни на содержимое остальных аккумуляторов, ни на биты состояния.

Операция добавления константы предназначена выполнять сложение указанной в инструкции константы с содержимым аккумулятора 1.

Вы можете определить константу как шестнадцатеричную константу формата byte или как десятичную константу формата word или как десятичную константу формата duobleword. Если необходимо прибавить константу формата word, используя тип DINT, добавьте к описанию константы префикс L#. Если десятичная константа по величине превышает разрешенный диапазон для типа INT, вычисление (добавление константы) автоматически выполняется как для типа данных DINT.

Вы можете записать десятичное число со знаком минус - таким образом можно выполнить операцию вычитания константы из значения в аккумуляторе. Перед добавлением байтовой константы она преобразуется в число формата INT со знаком.

Как и для арифметических функций, с данными типа INT операция добавления байтовой константы или константы формата word влияет только на младшее слово аккумулятора 1 и не влияет на старшее слово аккумулятора. Если разрешенный диапазон для типа INT нарушается, то бит 15 (бит знака) переписывается.

Операция добавления константы формата word обрабатывает все 32 бита аккумулятора, что соответствует операции вычисления для данных типа DINT.

Указанные операции добавления константы выполняются вне всякой связи с какими-либо условиями.

Пример операции добавления константы:

L AddValue1

+ B#16#21

T AddResult1

В примере значение переменной AddValue1 увеличивается на 33 и передается в переменную AddResult1.

L AddValue2

+ -33

T AddResult2

В этом примере значение переменной AddValue2 уменьшается на 33 и сохраняется в переменной AddResult2.

L AddValue3

+ L#-1

T AddResult3

В последнем примере значение переменной AddValue3 уменьшается на 1 и сохраняется в переменной AddResult3. Операция вычитания как при вычислениях с данными типа DINT.

4.6.4. Операции декрементирования и инкрементирования

Вы можете запрограммировать операции декрементирования и инкрементирования в соответствии со следующей общей схемой:

загрузка адреса Address,

декрементирование/ инкрементирование,
передача результата Result.

DEC n // Декрементирование
INC n // Инкрементирование

Операции декрементирования и инкрементирования меняют значение в аккумуляторе 1. Содержимое аккумулятора уменьшается (декрементируется) или увеличивается (инкрементируется) на число, определенное в параметре инструкции. Этот параметр может принимать значение в диапазоне от 0 до 255. При этом изменяется только значение в младшем байте аккумулятора. Старший байт аккумулятора остается без изменения. Операция инкрементирования выполняется таким образом, что если значение в результате инкрементирования превышает величину 255, то расчет значения начинается с начала (с 0), а операция декрементирования выполняется так, что если значение в результате декрементирования становится меньше 0, то расчет значения начинается с максимально возможного значения (с 255).

Операции декрементирования и инкрементирования выполняются независимо от значения RLO. Эти операции выполняются при появлении соответствующих инструкций и не влияют ни на RLO, ни на биты состояния.

Примеры операций декрементирования и инкрементирования:

L IncValue
INC 5
T IncResult

В примере значение переменной IncValue увеличивается на 5 и передается в переменную IncResult.

L DecValue
DEC 7
T DecResult

В примере значение переменной DecValue уменьшается на 7 и сохраняется в переменной DecResult.

4.6.5. Математические функции

К математическим функциям относятся следующие функции:

- синус (SIN), косинус (COS), тангенс (TAN);
- арксинус (ASIN), арккосинус (ACOS), арктангенс (ATAN) ;
- возведение в квадрат (SQR), извлечение квадратного корня (SQRT);
- экспонента (EXP), логарифм (LN).

Все математические функции обрабатывают числа в формате REAL. В зависимости от результата математические функции устанавливают биты состояния CC0, CC1, OV и OS (см. раздел "Биты состояния").

Математические функции в качестве входного значения используют число, находящееся в аккумуляторе 1, данное число обрабатывается в соответствии с инструкцией функции и вновь сохраняется в аккумуляторе 1.

Вы можете программировать математические функции в соответствии со следующей общей схемой:
загрузка адреса Address;
математическая функция;
передача (transfer) результата Result.

Математические функции изменяют только содержимое аккумулятора 1; содержимое всех остальных аккумуляторов остается неизменным. Математические функции всегда выполняются вне всякой связи с какими-либо условиями.

Математические функции выполняются в соответствии с правилами обработки чисел типа REAL, даже когда применяется абсолютная адресация и тип входного числа не описан.

В случае, когда аккумулятор 1 содержит некорректное действительное число (invalid REAL number) при выполнении математической функции, тогда функция возвращает некорректное результирующее значение (REAL) в аккумулятор 1 и биты состояния CC0, CC1, OV и OS устанавливает в "1".

Тригонометрические функции SIN (синус), COS (косинус) и TAN (тангенс) воспринимают действительное число (REAL) в аккумуляторе 1 как значение угла, выраженное в радианах.

Обычно для измерения углов применяются две единицы измерения: градусы с диапазоном измерения от 0° до 360° и радианы с диапазоном измерения от 0 до 2 π (где $\pi = +3.141593e+00$). Обе эти единицы пропорциональны друг другу и могут быть преобразованы одна в другую. Например, угол 90° в радианах имеет значение $\pi/2$ или $+1.570796e+00$.

В случаях, когда угол, выраженный в радианах, имеет величину, большую, чем 2 π или $+6.283185e+00$, то из этой величины вычитается 2 π или кратное 2 π , до тех пор, пока угол не станет меньше 2 π .

Пример:

Расчет реактивной мощности: $P_s = U \cdot I \cdot \sin(\varphi)$.

L PHI //Загрузка угла фи

SIN

L Current //Загрузка значения тока

*R

L Voltage //Загрузка значения напряжения

*R

T I_Power //Сохранение значения реактивной мощности

Функции ASIN (арксинус), ACOS (арккосинус) и ATAN (арктангенс) являются обратными по отношению к соответствующим тригонометрическим функциям, рассмотренным выше. Эти функции используют действительное число (REAL), находящееся в аккумуляторе 1, и возвращают значение угла, выраженное в радианах.

Если превышаетс разрешенный диапазон значений, то обратные тригонометрические функции возвращают некорректное действительное число и устанавливают биты состояния CC0, CC1, OV и OS в "1".

Пользователю доступны также еще несколько математических функций:

- SQR - возведение в квадрат,
- SQRT - извлечение квадратного корня,
- EXP - экспоненциальная функция по основанию e,
- LN - логарифмическая функция по основанию e (натуральный логарифм).

Функция SQR выполняет операцию возведения в квадрат содержимого аккумулятора 1.

Пример: расчет объема цилиндра: $V = r^2 \pi H$.

L Radius //Загрузка значения радиуса основания

SQR

L Height //Загрузка значения высоты

*R

L 3.141592 //Загрузка значения числа π

*R

T Volume //Сохранение значения объема

Функция SQRT выполняет операцию извлечения квадратного корня из содержимого аккумулятора 1. Если содержимое аккумулятора меньше нуля, то функция возвращают некорректное действительное число и устанавливают биты состояния CC0, CC1, OV и OS в "1". Если аккумулятор содержит "-0" (минус 0), функция возвращает "-0".

4.6.6. Функции преобразования

Функции преобразования преобразуют (конвертируют) тип данных, находящихся в аккумуляторе accumulator 1. На рис.4.9 представлен обзор функций преобразования.

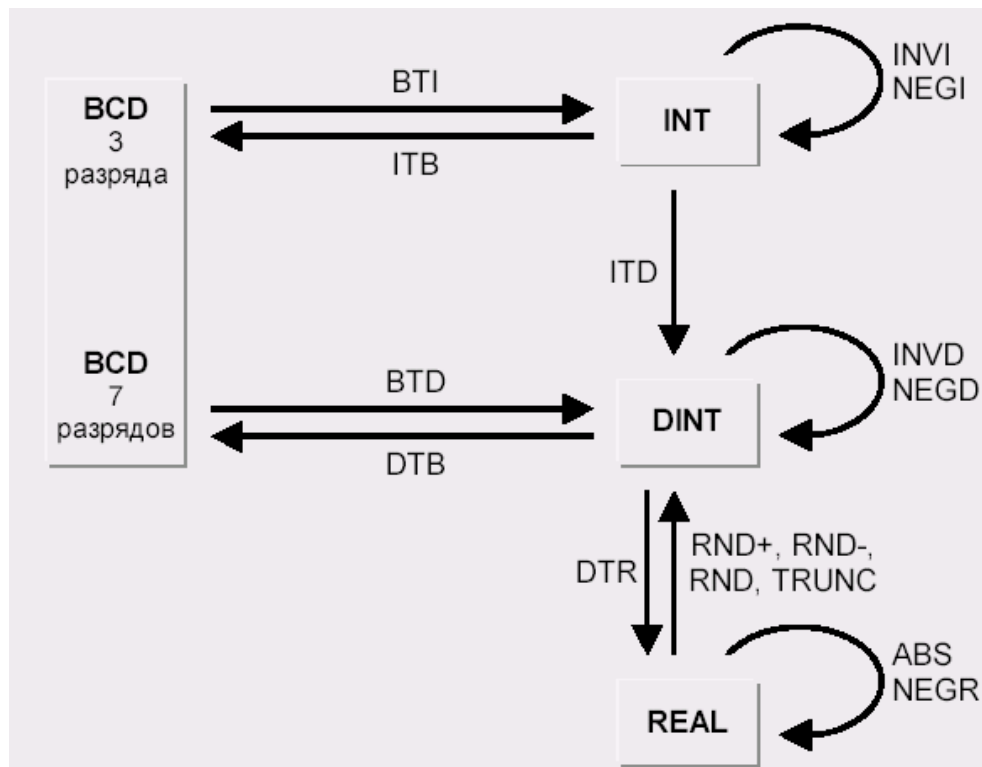


Рис.4.9 Обзор функций преобразования

Вы можете найти также подробную информацию о битовой структуре различных форматов данных в главе "Типы данных", а также информацию о том, как функции преобразования устанавливают биты состояния в разделе "Биты состояния".

Функции преобразования воздействуют только на данные, находящиеся в аккумуляторе 1. При этом отдельные функции преобразования воздействуют только на содержимое младшего слова в аккумуляторе 1 (биты с 0 по 15), другие функции воздействуют на содержимое аккумулятора в целом.

Вы можете программировать функции преобразования в соответствии со следующей общей схемой: загрузка содержимого адреса Address; функция преобразования; передача результата Result.

Функции преобразования конвертируют формат данных в соответствии с инструкцией функции, даже если используется абсолютная адресация и тип данных в аккумуляторе не был объявлен. Функции преобразования всегда выполняются вне всякой связи с какими-либо условиями.

Вы можете использовать содержимое аккумулятора 1 для нескольких последовательно выполняемых функций преобразования, и в этом поэтапном процессе преобразования нет необходимости использовать дополнительные ресурсы для временного хранения конвертированных значений.

Пример:

```
L BCD_Number
BTI          //BCD -> INT
ITD          //INT -> DINT
DTR          //DINT -> REAL
T REAL_Numbe
```

В данном примере BCD-число в три шага преобразуется в число формата REAL. Следующие функции обеспечивают выполнение преобразований чисел форматов INT и DINT:

- ITD - преобразует INT в DINT,
- ITB - преобразует INT в BCD,
- DTB - преобразует DINT в BCD,
- DTR - преобразует DINT в REAL.

Следующие функции обеспечивают выполнение преобразований чисел формата BCD:

- BTI - преобразует BCD в INT
- LTD - преобразует BCD в DINT

Функция BTI интерпретирует содержимое младшего слова в аккумуляторе 1 (биты с 0 по 15) как трехразрядное BCD-число и преобразует его в число типа INT. Три разряда числа в аккумуляторе 1

располагаются с выравниванием вправо и представляют собой значение десятичного числа. Знак располагается в битах с 12 по 15.

Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный. В процессе преобразования берется в расчет только состояние бита 15. Содержимое старшего слова (биты с 16 по 31) остается без изменений. При обнаружении "псевдотетрады" в исходном BCD-числе (численные значения с 10 по 15 (или с А по F) в шестнадцатеричном числе), CPU сообщает об ошибке назначения параметра ("a parameter assignment error") и вызывает организационный блок ОВ 121 (блок обработки синхронных ошибок).

Если блок ОВ 121 не запрограммирован, то CPU переходит в состояние STOP.

Функция преобразования ВТI не устанавливает битов состояния.

Функция ВТD интерпретирует содержимое аккумулятора 1 как семиразрядное BCD-число и преобразует его в число типа DINT. Семь разрядов числа в аккумуляторе accumulator 1 располагаются с выравниванием вправо и представляют собой значение десятичного числа. Биты с 28 по 31 содержат знак. Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный. В процессе преобразования берется в расчет только состояние бита 31.

При обнаружении "псевдотетрады" в исходном BCD-числе (численные значения с А по F в шестнадцатеричном числе), CPU ведёт себя как и в случае с функцией ВТI.

Существует несколько функций, обеспечивающих выполнение операций преобразования дробного числа формата REAL в число формата DINT (преобразование дробного числа в целое число). Эти функции отличаются друг от друга способом выполнения операции округления.

- RND+ - преобразование с округлением "вверх" до следующего целого,
- RND- - преобразование с округлением "вниз" до следующего целого,
- RND - преобразование с округлением до следующего целого числа,
- TRUNC - преобразование без округления (усечение).

В таблице 4.5 показано различие действий указанных функций преобразования чисел формата REAL в числа формата DINT. Диапазон от -1 до +1 был выбран для примера.

Функция RND+ интерпретирует содержимое аккумулятора 1 как число типа REAL и преобразует его в число формата DINT и возвращает целое число, которое больше исходного числа или равно ему.

Если исходное число в аккумуляторе выходит за пределы допустимого для формата DINT диапазона или исходное число имеет другой формат, отличный от формата REAL, то функция RND+ устанавливает биты состояния OV и OS. Преобразование в таком случае не может быть выполнено.

Таблица 4.5. Режимы округления функции преобразования чисел типа REAL

Входное значение		Результат			
REAL	DW#16#	RND	RND+	RND-	TRUNC
1.00000001	3F80 0001	1	2	1	1
1.00000000	3F80 0000	1	1	1	1
0.99999995	3F7F FFFF	1	1	0	0
0.50000005	3F00 0001	1	1	0	0
0.50000000	3F00 0000	0	1	0	0
0.49999996	3EFF FFFF	0	1	0	0
5.877476e-39	0080 0000	0	1	0	0
0.0	0000 0000	0	0	0	0
-5.877476e-39	8080 0000	0	0	-1	0
-0.49999996	BEFF FFFF	0	0	-1	0
-0.50000000	BF00 0000	0	0	-1	0
-0.50000005	BF00 0001	-1	0	-1	0
-0.99999995	BF7F FFFF	-1	0	-1	0
-1.00000000	BF80 0000	-1	-1	-1	-1
-1.00000001	BF80 0001	-1	-1	-2	-1

Существует еще несколько функций, обеспечивающих выполнение операций преобразования чисел:

- INVI - нахождение обратного кода двоичного числа формата INT,
- INV D - нахождение обратного кода двоичного числа формата DINT,
- NEGI - инвертирование числа формата INT,
- NEG D - инвертирование числа формата DINT,
- NEG D - инвертирование числа формата REAL.
- ABS - нахождение абсолютного значения числа формата REAL.

Функция INVI инвертирует содержимое младшего слова в аккумуляторе 1 (биты с 0 по 15) бит за битом. Функция каждый двоичный ноль заменяет двоичной единицей и наоборот, каждую единицу заменяет

нулем. Содержимое старшего слова (биты с 16 по 31) остается без изменений. Функция преобразования INVI не устанавливает битов состояния.

Функция INV D инвертирует содержимое в аккумуляторе 1 бит за битом. Функция каждый двоичный ноль заменяет двоичной единицей и наоборот, каждую единицу заменяет нулем. Функция преобразования INV D не устанавливает битов состояния.

Функция NEG I интерпретирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) как число типа INT и изменяет его знак. Эта операция идентична умножению на -1. При этом биты старшего слова в аккумуляторе accumulator 1 (биты с 16 по 31) не изменяют своего состояния. Функция преобразования NEG I устанавливает следующие биты состояния: CC0, CC1, OV и OS.

Функция NEG D интерпретирует содержимое аккумулятора accumulator 1 как число типа DINT и изменяет его знак. Эта операция идентична умножению на -1. При этом биты старшего слова в аккумуляторе accumulator 1 (биты с 16 по 31) не изменяют своего состояния. Функция преобразования NEG D устанавливает следующие биты состояния: CC0, CC1, OV и OS.

Функция NEG R интерпретирует содержимое аккумулятора 1 как число формата REAL и умножает это число на -1 (функция меняет знак мантиссы, даже если число в аккумуляторе является некорректным действительным числом). Функция NEG R не устанавливает битов состояния.

Функция ABS интерпретирует содержимое аккумулятора 1 как число формата REAL и формирует для этого числа абсолютное значение, устанавливая знак числа в значение "0" (функция устанавливает знак мантиссы в значение "0", даже если число в аккумуляторе является некорректным действительным числом). Функция ABS не устанавливает битов состояния.

4.7. Функции сдвига

Функции сдвига сдвигают влево или вправо содержимое аккумулятора 1 бит за битом. В таблице 4.6 представлен обзор доступных пользователю функций сдвига.

Таблица 4.6. Обзор функций сдвига

Функции сдвига	Word (Слово)		Duobleword (Двойное слово)	
	Число позиций как параметр	Число позиций в Accum2	Число позиций как параметр	Число позиций в Accum2
Сдвиг влево (Shift left)	SLW n	SLW	SLD n	SLD
Сдвиг вправо (Shift right)	SRW n	SRW	SRD n	SRD
Сдвиг со знаком вправо (Shift with sign)	SSI n	SSI	SSD n	SSD
Циклический сдвиг влево (Rotate left)	-	-	RLD n	RLD
Циклический сдвиг вправо (Rotate right)	-	-	RRD n	RRD
Сдвиг влево на 1 через бит CC1			RLDA ¹⁾	
Сдвиг вправо на 1 через бит CC1			RRDA ¹⁾	

¹⁾ Без параметров, как при смещении только одного бита

4.7.1. Выполнение функций сдвига

Функции сдвига позволяют побитно сдвигать влево или вправо данные, находящиеся в аккумуляторе 1. При этом в зависимости от функции сдвига аккумулятор содержит либо слово (word), либо двойное слово (duobleword). Биты, сдвигаемые за пределы слова (двойного слова), либо теряются (при операциях сдвига), либо переносятся на другую сторону двойного слова (при операциях циклического сдвига). Функции сдвига не влияют на содержимое других аккумуляторов.

Функции сдвига всегда выполняются вне всякой связи с какими-либо условиями. Они воздействуют только на содержимое аккумулятора 1. При этом функции не влияют на результат логической операции (RLO).

Параметры для функции сдвига могут быть заданы следующими двумя путями:

- с числом позиций, заданным в аккумуляторе 2;

- с числом позиций, заданным в инструкции с помощью параметра.

Вы можете программировать операции двумя способами в соответствии со следующими общими схемами:

загрузка числа позиций Number_of_positions;

загрузка содержимого адреса Address;

функция сдвига;

передача результата Result;

либо -

загрузка содержимого адреса Address;

функция сдвига с параметром "число позиций";

передача результата Result.

Функции сдвига устанавливают бит состояния CC0 в "0", а бит CC1 устанавливают в состояние, в котором находился последний перемещенный бит. Биты состояния проверяются с помощью двоичного опроса или в операциях перехода в соответствии с описанием в разделах "Биты состояния" и "Функции перехода".

Функции сдвига для данных типа Word воздействуют только на содержимое младшего слова в аккумуляторе 1 (биты с 0 по 15). Содержимое старшего слова аккумулятора остается неизменным.

Функция сдвига с битом состояния CC1 смещает содержимое аккумулятора 1 на одну битовую позицию.

Рассмотрим примеры применения функций сдвига. Значение в слове MW 130 сдвигается на 4 позиции влево и сохраняется в слове MW 132. Здесь число позиций задано параметром в инструкции.

```
L MW 130
```

```
SLW 4
```

```
T MW 132
```

Значение в переменной "ShiftOn" сдвигается вправо на "ShiftPos" позиций и сохраняется в переменной "ShiftOff". Здесь число позиций задано посредством аккумулятора 2.

```
L "Global_DB".ShiftPos
```

```
L "Global_DB".ShiftOn
```

```
SRD
```

```
T "Global_DB".ShiftOff
```

Сдвиг переменной #Actval со знаком вправо на 2 позиции и передача в переменную #Display.

```
L #Actval
```

```
SSI 2
```

```
T #Display
```

Вы можете выполнять операцию сдвига содержимого аккумулятора 1 любое необходимое число раз, например:

```
L Value1
```

```
SSD 4
```

```
SLD 2
```

```
T Result1
```

В данном примере в результате выполнения программы содержимое аккумулятора 1 будет сдвинуто со знаком вправо на 2 позиции при этом два правых бита будут сброшены в состояние "0".

4.7.2. Циклический сдвиг

Циклический сдвиг влево на n разрядов выполняется командой: RLD n , а командой - RLD выполняется циклический сдвиг влево на количество разрядов, указанное в аккумуляторе 2

Операция циклического сдвига RLD позволяет бит за битом сдвигать влево данные, находящиеся во всех битах аккумулятора 1. При этом вновь освобожденный при сдвиге содержимого разряд заполняется значением бита, который был "вытолкнут" из аккумулятора последним.

Число позиций, на которое выполняется операция циклического сдвига, может быть указано в параметре инструкции RLD или может быть загружено в аккумулятор 2, в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция циклического сдвига не выполняется (нет операции, т.е. NOP); если число позиций равно 32, то содержимое аккумулятора 1 не изменяется, а бит состояния CC1 принимает состояние бита, перемещенного последним (бит 0). Если указанное число позиций равно 33, то содержимое аккумулятора 1 смещается на 1 разряд, если указанное число позиций равно 34, то содержимое смещается на 2 позиции и так далее.

Циклический сдвиг вправо выполняется аналогично сдвигу влево командами: RRD n и RRD.

4.8. Логические функции для слов данных

Логические функции для слов данных (Word Logic) позволяют побитно комбинировать значение, находящееся в аккумуляторе 1, с содержимым аккумулятора 2. Результат выполнения операции сохраняется в аккумуляторе 1.

Логические функции могут выполняться как для данных формата Word (слов), так и для данных формата Duobleword (двойных слов).

4.8.1. Выполнение логических операций для слов данных

Вы можете программировать операции двумя способами в соответствии со следующей схемой:

загрузка содержимого адреса Address1;

загрузка содержимого адреса Address2;

логическая операция для слова данных без константы;

передача результата Result.

Логические функции для слов данных формируют результат операции бит за битом таким же образом, как и функции двоичной логики, описанные в главе "Двоичные логические операции".

Логическая функция комбинирует бит 0 аккумулятора 1 с битом 0 аккумулятора 2. Результат сохраняется в бите 0 аккумулятора 1. Таким же образом эта же логическая операция выполняется для битов 2, битов 3 и так далее вплоть до старшего бита операндов (до бита 15 в операциях для слов и до бита 31 в операциях для двойных слов). Содержимое аккумулятора 2 при этом остается неизменным.

Пример:

L MW 142 //Адрес 1

L MW 144 //Адрес 2

AW //Логическая операция «поразрядное и»

T MW 146 //Результат

В этом примере логическая функция выполняется с данными формата слово (Word).

Логические функции, выполняемые с данными формата слово (Word), воздействуют только на данные, находящиеся в младшем слове аккумулятора 1 (т.е. в битах с 0 по 15). При этом биты, находящиеся в старшем слове аккумулятора (биты с 16 по 31), остаются без изменения.

После завершения логической операции для слов данных Вы можете немедленно перейти к выполнению следующей логической операции (загрузив предварительно данные из соответствующего адреса) без необходимости сохранения промежуточного результата (например, в области локальных данных). Аккумуляторы сами служат для временного хранения данных.

Пример:

L Value1

L Value2

AW

L Value3

OW

T Result1

В вышеуказанном примере результат выполнения инструкции AW содержится в аккумуляторе 1. Во время загрузки значения Value3 этот результат смещается в аккумулятор 2. Теперь оба значения могут быть обработаны в соответствии с инструкцией OW.

Для операции поразрядного И служат команды:

AW - логическая операция И для данных формата Word (слово) в ассум1 и ассум,

AD - логическая операция И для данных формата Duobleword (двойное слово) в ассум1 и ассум2.

Логическая функция для обработки двух чисел в соответствии с логикой AND (И) позволяет последовательно бит за битом комбинировать данные, находящиеся в аккумуляторе 1, с соответствующими битами аккумулятора 2. Бит, находящийся в слове результата, будет установлен в состояние "1" только в случае, если оба сравниваемых бита исходных данных имеют значение "1". Так как те биты в аккумуляторе 2, которые имеют значение "0", сбрасывают в "0" соответствующие биты слова результата вне зависимости от состояния соответствующих битов в аккумуляторе 1, то говорят, что эти биты "маскированы" ("masked"). Этот, так называемый, эффект маскирования битов и является основным назначением логических операций (И) для слов данных.

Логическая функция для обработки двух чисел в соответствии с логикой Exclusive OR (Исключающее ИЛИ) позволяет последовательно бит за битом комбинировать данные, находящиеся в аккумуляторе 1, с соответствующими битами аккумулятора 2. Бит, находящийся в слове результата, будет установлен в состояние "1" только в случае, если только один из сравниваемых битов исходных данных имеет значение "1". Если некоторый бит в аккумуляторе 2 имеет значение "1", то соответствующий бит слова результата в соответствии с логикой функции будет иметь инвертированное значение по

отношению к предыдущему значению этого бита в аккумуляторе 1. В результате те биты в аккумуляторе 2, которые имеют значение "1", устанавливают в "1" соответствующие биты слова результата вне зависимости от состояния соответствующих битов в аккумуляторе 1, то говорят, что эти биты "маскированы" ("masked").

5. Управление выполнением программы

STEP 7 обеспечивает пользователя различными средствами для управления ходом выполнения программы. Вы можете выходить из выполняемой линейной программы внутри блока, Вы можете также создать определенную структуру программы посредством вызовов параметризуемых блоков. Вы можете воздействовать на выполнение программы, исходя из текущих значений параметров, рассчитанных во время работы программы, в зависимости параметров процесса или в связи с состоянием установки.

Биты состояния (status bits) призваны обеспечивать информацией, зависящей от результатов выполнения арифметических и математических функций, а также от возникающих ошибок (например, из-за нарушения границ диапазона допустимых значений при расчетах). Вы можете использовать состояния сигналов этих битов в Вашей программе непосредственно, проверяя их с помощью двоичных логических операций.

Функции перехода (jump functions) могут использоваться для организации ветвления программы пользователя. Функции перехода могут быть либо безусловные, либо условные, (по условию определяемому отдельным битом состояния, результатом логической операции RLO или двоичным результатом). В STL можно легко запрограммировать переходы в расчетные точки программы (распределитель переходов) или выполнить программные циклы (циклический переход).

Главное управляющее реле (Master Control Relay - MCR) обеспечивает еще один способ управления выполнением программы. Изначально разработанный для систем управления с помощью реле, язык STL обеспечивает программную (компьютерную) версию программируемого способа управления.

Функции блоков (block functions) позволяют пользователю структурировать программу. Вы можете снова и снова использовать функции и функциональные блоки, определяя **параметры блоков**.

5.1. Биты состояния

Биты состояния - это двоичные флаги (индикаторные биты). CPU использует их для управления двоичными логическими операциями. CPU устанавливает биты состояния при выполнении операций обработки чисел. Вы можете проверять состояние этих битов (например, для выяснения результата выполнения вычислений); Вы можете также влиять на отдельные биты. Биты состояния скомпонованы в слово, в "слово состояния" ("status word"). В таблице 5.1 представлены биты состояния, доступные при STL-программировании.

Таблица 5.1. Биты состояния

Бит	Двоичные флаги (binary flags)	
0	/FC	Первичный опрос (first check)
1	RLO	Результат логической операции
2	STA	Состояние (статус - "status")
3	OR	Бит состояния OR (OR status bit)
4	OS	Для сохранения информации о переполнении (stored overflow)
5	OV	Переполнение (overflow)
6	CC0	Условный код (condition code)
7	CC1	Условный код (condition code)
8	BR	Двоичный результат (binary result)

Первый столбец показывает номер бита в слове состояния. CPU использует "двоичные флаги" ("binary flags") для управления двоичными функциями; "числовые флаги" ("digital flags") используются прежде всего для индикации результатов арифметических и математических функций.

Первичный опрос (first check) - бит состояния /FC, который управляет двоичными логическими операциями внутри логического устройства управления. Двоичный логический шаг всегда начинается с двоичной инструкции проверки (первичного опроса) при /FC = "0", как показано в описании двоичных

логических операций. Первичный опрос устанавливает /FC в состояние "1". Двоичный логический шаг заканчивается присвоением двоичного значения, или условным переходом, или изменением блока. Это сбрасывает бит состояния: /FC = "0". Следующая двоичная проверка (опрос) начинается с новой двоичной логической функции.

Результат логической операции (RLO) является промежуточным буфером в двоичных логических операциях. При первичном опросе CPU передает результат опроса в RLO, комбинирует результат опроса с хранящимся в RLO значением при каждом последующем опросе и затем сохраняет результат в RLO (как описано в главе "Двоичные логические операции"). Вы можете также устанавливать, сбрасывать или инвертировать значение в RLO непосредственно или сохранять его в BR. Функции счетчика, таймера и операции с памятью управляются с использованием RLO, как и функции перехода.

Состояние бита **STA** соответствует состоянию сигнала указанного двоичного разряда (двоичного адреса) или проверяемого "условного бита" для двоичной логической операции (A, AN, O, ON, X, XN). В случае операций с памятью (S, R, =) значение бита состояния STA соответствует записанному в память значению или (в случае отсутствия операции записи в память, например, при RLO = "0" или когда главное управляющее реле MCR активно) значение бита STA соответствует значению адресованного (но неизмененного) двоичного адреса. При проверке наличия фронта сигнала FP или FN значение RLO до операции проверки фронта сохраняется в бите состояния STA. Все остальные двоичные операции устанавливают STA (STA = "1"), как и переходы, зависящие от состояния двоичного флага: JC, JCN, JBI, JNBI (исключение: CLR сбрасывает бит STA: STA = "0"). Бит состояния STA не влияет на обработку операторов STL-операторов. Состояние этого бита отображается тестовыми функциями с помощью программатора PG (такой функцией, например, как функция отображения состояния программы "program status"). Таким образом, Вы можете использовать этот бит состояния для трассировки двоичных логических последовательностей или в целях отладки программы.

Бит OR (OR status bit) сохраняет результат выполненной (имеется в виду, что условие функции выполнено) двоичной логической операции И ("1") и показывает последующей операции ИЛИ, что результат уже зафиксирован (при комбинированной операции глобального О (ИЛИ) внутри операции перед её выполнением). Все остальные двоичные операторы сбрасывают бит OR status.

Рассмотрим пример влияния битов состояния.

```
...
= M 10.0
A I 4.0
A I 4.1
O          // бит OR status устанавливается равным RLO
O I 4.2    // бит OR status сбрасывается
ON I 4.3
= Q 8.0
R Q 8.1
S Q 8.2
```

Бит состояния **OV (Overflow)** индицирует факт превышения диапазона допустимых численных значений (**переполнения**) или факт использования некорректных действительных (REAL) чисел. На состояние бита OV влияют следующие функции: арифметические, математические, отдельные функции преобразования, функции сравнения действительных (REAL) чисел. Вы можете проверить состояние бита OV с помощью операций проверки или посредством оператора перехода JO.

Бит состояния **OS (Stored overflow)** фиксирует факт превышения (дублирует и сохраняет установленное состояние бита OV). Когда CPU устанавливает бит состояния OV, он также устанавливает бит состояния OS. При этом, если бит состояния OV в дальнейшем может быть сброшен при выполнении соответствующей операции, то бит OS сохранит свое состояние, т.е. информацию о состоявшемся факте превышения. Это позволит Вам выполнять альтернативную проверку бита состояния, в том числе и отложенную во времени (в более поздней точке программы), для определения факта превышения диапазона допустимых численных значений или факта использования некорректных действительных (REAL) чисел.

Вы можете проверять состояние бита OS посредством операторов проверки (опроса) или с помощью оператора перехода JOS. Оператор перехода JOS или смена блока сбрасывает бит состояния OS.

Биты состояния **CC0, CC1** (условные биты) призваны обеспечивать информацией о результатах выполнения функций сравнения, арифметических и математических функций, логических операций для слов данных или о состоянии "вытолкнутых" битов в случае выполнения операций сдвига. Вы можете

проверять состояние всех числовых флагов с помощью операторов перехода или посредством операторов проверки (опроса) (см. далее).

Бит **BR** (двоичный результат) помогает реализовать механизм EN/ENO для вызовов блоков (в сочетании с графическими языками). Вы можете также установить или сбросить бит состояния BR или проверить его состояние с помощью операторов проверки (опроса) или посредством операторов перехода.

Слово статуса (status word) содержит в себе все биты состояния. Вы можете загружать слово состояния в аккумулятор 1, а также считать его значение из этого аккумулятора:

L STW //загрузить слово статуса в аккумулятор 1

T STW //записать в слово статуса

Вы можете использовать слово статуса для проверки битов состояния или для их установки в соответствии с Вашими требованиями. Таким образом, Вы можете сохранить текущее значение слова статуса или начать выполнение программного блока с требуемыми значениями битов состояния. Надо учитывать, что S7-300 CPU (кроме CPU 318) не загружают биты /FC, STA и OR в аккумулятор 1; аккумулятор в соответствующих этим битам позициях содержит "0".

5.1.1. Описание битов состояния

Функции обработки чисел влияют на биты состояния CC0, CC1, OV и OS. Отдельные операторы STL влияют на биты состояния RLO и BR.

Арифметические функции при использовании чисел форматов INT и DINT могут устанавливать все числовые флаги ("digital flags", биты состояния). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). Деление на ноль приводит к установлению в состояние "1" всех "числовых битов состояния" ("digital status bits").

Арифметические функции при использовании чисел формата REAL могут устанавливать все "числовые биты состояния" ("digital status bits"). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). При обработке некорректного действительного (REAL) числа происходит установка в состояние "1" всех числовых битов состояния.

Действительное (REAL) число относится к "ненормированным" (или "ненормализованным" - "denormalized"), если оно представляется с уменьшенной точностью. Абсолютное значение "ненормированного" ("denormalized") числа формата REAL меньше чем $1.175494 \cdot 10^{-38}$). S7-300 интерпретируют ненормированные числа формата REAL как числа, равные нулю (см Таб. 5.2).

Таблица 5.2. Значения числа формата REAL

Описание	Значение e	Мантисса f	Значение	CC1	CC0	OV	OS
Нет представл.	255	<>0	[qNaN]	1	1	1	1
Переполнение	255	0	$>2^{127}$ или $<-2^{127}$	1 0	0 1	1 1	1 1
Нормализованное число	1.. 254	любая	(1.f) 2^{e-127} (-1.f) 2^{e-127}	1 0	0 1	0 0	- -
Денормализованное число	0	<>0	(0.f) 2^{-126} (- 0.f) 2^{-126}	0 0	0 0	1 1	1 1
Нуль	0	0	+0	0	0	0	-

Функции инвертирования влияют на все "числовые биты состояния". Кроме того, следующие функции преобразования устанавливают биты состояния OV и OS в случае появления ошибок - выход за пределы диапазона допустимых значений и обработка некорректного числа формата REAL:

- ITB и DTB - преобразование числа формата INT в формат BCD,
- RND+, RND-, RND, TRUNK - преобразование числа формата REAL в формат DINT.

Функции сравнения устанавливают биты состояния CC0 и CC1, которые устанавливаются независимо от типа выполняемой функции сравнения; зависят они только от отношения между двумя значениями, указанными в выражении функции сравнения. Функция сравнения чисел формата REAL производит проверку корректности данных формата REAL.

Логические операции для слов и функции сдвига устанавливают биты состояния CC0 и CC1. Бит OV находится в сброшенном состоянии ("0").

5.1.2. Проверка битов состояния

Биты состояния RLO и BR и все числовые флаги могут быть проверены с помощью операций двоичного опроса и функций перехода. Также можно обрабатывать биты состояния после загрузки в аккумулятор 1 слова состояния.

Вы можете использовать все операторы опроса (проверки), описанные в главе "Двоичные логические операции", для проверки числовых флагов (digital flags) и двоичного результата BR:

- >0 - результат больше 0 $[(CC0=0) \& (CC1=1)]$,
- >=0 - результат больше или равен 0 $(CC0=0)$
- <0 - результат меньше 0 $[(CC0=1) \& (CC1=0)]$,
- <=0 - результат меньше или равен 0 $(CC1=0)$,
- <>0 - результат не равен 0 $[(CC0=1) \& (CC1=0)] \vee [(CC0=0) \& (CC1=1)]$,
- ==0 - результат равен 0 $[(CC0=0) \& (CC1=0)]$,
- UO - результат не верен $[(CC0=1) \& (CC1=1)]$,
- OV - переполнение $(OV=1)$,
- OS - сохраненное переполнение $(OS=1)$,
- BR - двоичный результат.

При этом принципы использования операторов опроса (проверки) такие же, как и при проверке, например, входов.

Вы можете проверять биты состояния RLO и BR, все комбинации CC0 и CC1, а также биты состояния OV и OS с помощью соответствующих функций перехода. В главе "Функции перехода" содержится подробное описание.

Если результат вычисления выходит за пределы диапазона допустимых значений для определенного типа данных, то бит состояния OV устанавливается в "1"; одновременно устанавливается в "1" бит состояния OS (бит для сохранения информации о том, что имело место нарушение границы диапазона допустимых значений).

Если результат следующей далее функции (например, в цепочке вычислений) находится в пределах допустимых значений, флаг OV будет сброшен. Однако, при этом флаг OS останется установленным, так что тот факт, что имело место превышения границ разрешенного диапазона значений, останется сохраненным, при этом он может быть обнаружен с помощью проверки (опроса) в конце вычислений. Флаг OS не может быть сброшен, пока не встретится функция перехода JOS или не сменится блок (т.е., пока не произойдет новый вызов блока или пока не закончится обработка текущего блока). Далее представлены два варианта программного решения задачи проверки состояния флагов для определения того факта, что имело место превышение пределов диапазона допустимых значений - в первой программе выполняется двоичный опрос (проверка) флагов, а во второй - выполняется проверка флагов с помощью условных переходов.

// Двоичный опрос

L Value1

L Value2

+I;

A OV // Отдельная проверка

= Status1

L Value3

+I

A OV // Отдельная проверка

= Status2

L Value4

+I;

A OS // Отдельная проверка

= Status_overall

T Result

// Функции перехода

L Value1

L Value2

+I

JO ST1 // Отдельная проверка

L Value3

```

+I
JO ST2 // Отдельная проверка
L Value4
+I;
JOS STOV // Общая проверка
T Result

```

Как видно из последних примеров, опрос (проверку) флагов можно производить либо после каждой операции вычисления (опрос бита состояния OV), либо в конце цепочки вычислений (опрос бита состояния OS).

5.1.3. Использование двоичного результата (бита BR)

В Вашем распоряжении двоичный результат BR как дополнительный бит RLO. Тем не менее, для индикации ошибок при обработке блоков (как это используется в системных блоках SFB и SFC и некоторых стандартных блоках) Вы можете использовать BR как флаг групповой ошибки, даже если Вы используете исключительно программирование на STL.

Вы можете использовать двоичный результат в качестве сигнального бита для сообщения о групповой ошибке в блоках. Если блок обрабатывается без ошибок, бит BR устанавливается в состояние "1", иначе - BR сбрасывается в состояние "0".

5.2. Команды перехода

С помощью команд перехода (jump functions) Вы можете прервать линейное выполнение программы и продолжить ее выполнение в другой точке блока. Такое ветвление программы может быть организовано либо с проверкой выполнения определенного условия, либо без проверки каких-либо условий (соответственно в программе - либо условный, либо безусловный переход).

Распределитель переходов (распределение переходов с проверкой параметра) и циклический переход также доступны пользователю как особые формы функций перехода.

В STL используются следующие операторы перехода:

```

JU метка // Безусловный переход
JC метка // Переход, если RLO = "1"
JCN метка // Переход, если RLO = "0"
JCB метка // Переход, если RLO = "1", и сохранение RLO
JNB метка // Переход, если RLO = "0", и сохранение RLO
JBI метка // Переход, если BR = "1"
JNBI метка // Переход, если BR = "0"
JZ метка // Переход, если результат равен "0"
JN метка // Переход, если результат не равен "0"
JP метка // Переход, если результат больше "0"
JPZ метка // Переход, если результат больше или равен "0"
JM метка // Переход, если результат меньше "0"
JMP метка // Переход, если результат меньше или равен "0"
JUO метка // Переход, если результат неверен
JO метка // Переход при переполнении (при проверке бита OV)
JOS метка // Переход при переполнении (при проверке бита OS)
JL метка // Переход в распределителе переходов
LOOP метка // Переход циклический

```

Таблица 5.3. Проверка битов состояния с помощью функций перехода

RLO	BR	CC0	CC1	OV	OS	Функции перехода
1	-	-	-	-	-	JC, JCB
0	-	-	-	-	-	JCN, JNB
-	1	-	-	-	-	JB
-	0	-	-	-	-	JNB
-	-	0	0	-	-	JZ, JMZ, JPZ
-	-	0	1	-	-	JN, JP, JPZ
-	-	1	0	-	-	JN, JM, JMZ
-	-	1	1	-	-	JUO
-	-	-	-	1	-	JO
-	-	-	-	-	1	JOS

5.2.1. Программирование функций перехода

Инструкция для каждой функции перехода содержит оператор перехода, определяющий проверяемое условие, и метку перехода, которая в свою очередь показывает, в какой точке программы следует продолжать ее обработку в случае, когда условие для перехода выполняется. Метка перехода может содержать до 4 символов алфавита и числового ряда, а также символ подчеркивания. Метка перехода не может начинаться с цифры. После метки перехода должно следовать двоеточие. Метка перехода указывает на выражение (строку), которая должна обрабатываться после выполнения операции перехода. Регистр клавиатуры учитывается в имени метки.

Переход может выполняться как вперед (в направлении выполнения программы, т.е., в направлении увеличения номеров строк), так и назад. Переход может выполняться только внутри блока; то есть, назначение точек перехода должно задаваться в том же блоке, где находятся выражения с оператором перехода. Разбиение блока на сегменты не влияет на функцию перехода. Точки перехода должны иметь уникальный идентификатор, что означает, что Вы можете назначать любую метку перехода в блоке, но только один раз с таким именем. К одной точке (метке) переход может осуществляться из нескольких точек программы блока.

При программировании на языке STL идентификаторы меток сохраняются в соответствующей неисполняемой части блока на носителях данных программатора. Только величина перехода хранится в рабочей памяти CPU (в скомпилированном блоке). По этой причине изменения в программе блока, сделанные в интерактивном режиме в CPU, должны также быть сохранены и на дисках программатора для сохранения исходных назначений. Если обновления программы не сделаны или если блоки были перенесены из CPU в программатор, то соответствующие неисполняемые части блоков переписываются или уничтожаются. Затем редактор сгенерирует свои собственные назначения меток (M001, M002 и т.д.) на экране или в распечатке.

Оператор безусловного перехода JU выполняется всегда, т.е. не зависит ни от каких условий. Команда JU прерывает последовательное выполнение программы и продолжает это выполнение с другой точки программы - с метки перехода, указанной в инструкции. Оператор безусловного перехода не влияет на биты состояния. Если встречаются операторы проверки (опроса), такие, например, как A I, O I и т.д., располагающиеся непосредственно перед командой перехода и сразу после метки перехода, то они воспринимаются как отдельные логические операции.

5.2.2. Функции перехода в зависимости от состояния RLO и BR

Переход в программе может быть сделан зависящим от состояния сигналов битов RLO и BR (см. табл. 5.4). Кроме того, одновременно можно сохранить результаты проверки битов состояния RLO и BR.

Таблица 5.4. Функции перехода с RLO и BR

RLO	BR	Выполняемые переходы	
"1"	-	JC	Переход, если RLO = "1"
"1"	--> "1"	JCB	Переход, если RLO = "1" и сохранить RLO
"0"	-	JCN	Переход, если RLO = "0"
"0"	--> "0"	JNB	Переход, если RLO = "0" и сохранить RLO
-	"1"	JBI	Переход, если BR = "1"
-	"0"	JNBI	Переход, если BR = "0"

5.2.3. Функции перехода в зависимости от состояния CC0 и CC1

Переход в программе может быть сделан зависящим от состояния сигналов битов CC0 и CC1 (см. табл. 5.5). Это позволяет, например, проверять, положителен ли результат вычисления, равен ли он нулю или меньше нуля (отрицательный результат). Подробную информацию о том, как устанавливаются биты состояния CC0 и CC1.

Таблица 5.5. Функции перехода с CC0 и CC1

CC0	CC1	Выполняемые переходы	
0	0	JZ	Переход, если результат = 0
		JMZ	Переход, если результат = 0 или < 0
		JPZ	Переход, если результат = 0 или > 0
1	0	JM	Переход, если результат < 0
		JMZ	Переход, если результат = 0 или < 0
		JN	Переход, если результат <> 0
0	1	JP	Переход, если результат > 0
		JPZ	Переход, если результат = 0 или > 0
		JN	Переход, если результат <> 0
1	1	JUO	Переход, если результат неверен

Операторы перехода, зависящие от CC0 и CC1, не изменяют никаких битов состояния. Если переход выполнен, значение RLO остается при продолжении программы со строки с меткой перехода и может

использоваться далее в программе (без изменения в /FC). Двоичный опрос является другим способом проверки битов состояния.

5.2.4. Переход в зависимости от состояния OV и OS

Переход в программе может быть сделан зависящим от состояния сигналов битов OV и OS. Это позволяет проверять, находится ли все еще результат вычисления в диапазоне допустимых значений. Подробную информацию о том, как устанавливаются биты состояния OV и OS, Вы можете найти в разделе "Биты состояния".

Команда перехода JO выполняется только в том случае, если OV = "1". Это случается, если результат вычисления выходит за пределы допустимых значений в предыдущей операции. Бит состояния OV может быть установлен такими функциями, как:

- арифметические функции;
- математические функции;
- функции инвертирования числа;
- функции сравнения для чисел формата REAL;
- функции преобразования чисел форматов INT/DINT в формат BCD и чисел формата REAL в формат DINT. Если бит состояния OV = "0", то условие для перехода JO не выполняется и выполнение программы продолжается со следующей инструкции.

Если выполняется цепочка последовательных вычислений, то состояние бита OV должно проверяться после выполнения каждого вычисления, так как бит состояния OV вновь будет сброшен после выполнения операции вычисления, результат которой будет находиться в диапазоне допустимых значений.

Бит состояния OS может быть опрошен после выполнения цепочки вычислений для проверки, не вышел ли за пределы допустимых значений результат одного из вычислений в рассматриваемой последовательности операций.

Команда перехода JOS выполняется только в том случае, если OS = "1". Это происходит во всех случаях, когда результат вычисления выходит за пределы допустимых значений, что вызывает установку бита состояния OV (см. выше). В отличие от OV, бит OS сохраняет свое состояние, даже если после его установки результат одной из последующих операций будет в диапазоне допустимых значений. Бит состояния OS может быть сброшен при следующих обстоятельствах:

- при вызове блока и при завершении блока;
- при выполнении перехода JOS.

Если бит состояния OS = "0", то условие для перехода JOS не выполняется и выполнение программы продолжается со следующей инструкции.

5.2.5. Распределитель переходов

Распределитель переходов (Jump Distributor) JL позволяет определять (вычислять) переходы в разделе программы в блоке к разным точкам, отмеченным метками. Команда JL работает вместе с набором операций перехода JU.

Последовательность выражений с операторами перехода JU следует сразу после функции JL и может содержать до 255 строк (входов). В инструкции после оператора JL следует метка перехода, указывающая на конец списка функций перехода JU (на первую инструкцию, следующую за набором функций перехода JU). Вы можете запрограммировать распределитель переходов в соответствии со следующей общей схемой:

```
L Number_of_positions //номер позиции
JL End
JU M0
JU M1
JU M2
...
JU Mx
End: ...
```

В этом примере переменная *Number_of_positions* содержит число, загружаемое в аккумулятор 1. Вслед за операцией загрузки следует распределитель переходов JL с меткой, указывающей на конец списка функций перехода JU. Номер перехода, который должен быть выполнен, должен содержаться в правой байте аккумулятора 1. Если этот аккумулятор содержит 0, то выполняется первая функция перехода. Если этот аккумулятор содержит 1, то выполняется вторая функция перехода и так далее. Если число в аккумуляторе 1 превышает размер списка операторов перехода, то происходит переход на конец списка (на первую инструкцию, следующую за набором функций перехода JU). JL не зависит ни от каких условий и

не изменяет битов состояния. При этом в списке операторов перехода допускаются только выражения с оператором JU, располагающиеся без пробелов. Назначая произвольные метки для этих операторов перехода, Вы должны придерживаться общих правил для меток.

5.2.6. Циклический переход

Команда циклического перехода (Loop Jump) LOOP позволяет просто реализовать в программе программные циклы. Функция LOOP интерпретирует число в правом слове (word) аккумулятора 1 как беззнаковое 16-разрядное число из диапазона от 0 до 65535.

При выполнении оператор LOOP сначала уменьшает содержимое аккумулятора 1 на 1. Если значение при этом не равно нулю, то выполняется переход к указанной метке. Если значение после уменьшения равно нулю, то выполняется следующее за телом цикла выражение. Значение в аккумуляторе 1, таким образом, соответствует числу циклов, которые должны быть выполнены. Вы должны сохранить это число в счетчике цикла. Вы можете использовать любое число (адрес) как счетчик цикла.

Вы можете запрограммировать циклический переход LOOP в соответствии со следующей общей схемой:

```
L Number           // загрузка числа циклов
Next: T Counter     // сохранение счётчика в переменной Counter
...                // тело цикла
L Counter           // загрузка счётчика
LOOP Next
```

В этом примере переменная *Number* содержит общее число циклов, которое необходимо выполнить. Переменная *Counter* содержит текущее число циклов, которое осталось выполнить.

При первом проходе цикла значению *Counter* задается число циклов, которое необходимо выполнить. В конце программного цикла содержимое переменной *Counter* загружается в аккумулятор и уменьшается с помощью оператора LOOP. Если после этого аккумулятор не содержит нуль, то выполняется переход к метке цикла (к метке Next). После этого происходит обновление переменной *Counter*.

Функция циклического перехода LOOP не изменяет битов состояния.

5.2.7. Null-операции

Поскольку в языке программирования STL метка не может помечать пустую строку, то для этого существуют Null-операции (нуль-операции). Они не создают никакого воздействия при выполнении программы. В STL есть операторы NOP 0, NOP 1 и BLD, используемые в качестве Null-операций (нуль-операций).

Вы можете использовать операторы NOP 0 (набор битов 16 x "0") и NOP 1 (набор битов 16 x "1") для использования в инструкциях, которые не выполняют никакого действия.

Примечание: необходимо отметить, что Null-операции (нуль-операции) занимают в памяти определенное пространство (2 байта) и для выполнения инструкции требуют определенного времени.

Пример: необходимо, чтобы в строке с меткой перехода присутствовала инструкция. Если необходимо использовать операцию перехода, но не нужно выполнять никаких операций в строке с меткой перехода, то Вы можете в этой строке использовать оператор NOP 0.

```
A I1.0
JC MXX1
...
MXX1: NOP 0
```

Вы можете ввести пустую строку для лучшей читаемости программы простым вводом строки комментария (пустой строки) (это не требует использования пространства памяти пользователя и не добавляет потерь в общее время выполнения программы, так как не содержит кода).

Редактор использует инструкцию отображения программы BLD nnn для включения информации декомпиляции в программу. Сами операторы BLD не отображаются в инкрементальном режиме.

5.3. Блоки

Если кодовый блок должен быть обработан, он должен быть "вызван", например, в организационном блоке OV 1. Вызов блока состоит из оператора вызова (CALL FC10, например) и списка параметров. Если вызываемый блок не имеет параметров, инструкция вызова не будет иметь списка параметров.

После выполнения оператора вызова CPU продолжает выполнение программы в вызываемом блоке. Программа блока обрабатывается, пока не встретится оператор окончания блока. По окончании вызванного блока CPU возвращается к выполнению программы в вызывающем блоке (в примере: OB 1); выполнение этой программы продолжается со следующего оператора после оператора вызова блока. Если завершается выполнение программы организационного блока, CPU передает управление операционной системе.

Информация, которая требуется CPU для возврата в вызывающий блок, сохраняется в стеке блоков (B stack). При каждом вызове блока в B-стеке генерируется новый элемент стека, который содержит адрес возврата, содержимое регистра данных и адрес стека локальных данных вызывающего блока. Если CPU переходит в состояние STOP в результате ошибки, Вы можете использовать программатор для того, чтобы увидеть в B-стеке информацию о том, какие блоки обрабатывались до момента возникновения ошибки.

Параметры блока являются интерфейсом данных для вызываемого блока. Рекомендуется избегать передачи данных посредством внутренних регистров (например, посредством аккумуляторов, адресных регистров, RLO), так как содержимое этих регистров может быть изменено при смене текущего блока (в результате "скрытых" операторов, инициированных редактором).

С помощью оператора CALL вызова блока Вы можете вызывать функциональные блоки FB, функции FC, системные функциональные блоки SFB, системные функции SFC. Оператор CALL является безусловным вызовом, что означает: заданный блок всегда вызывается и обрабатывается, несмотря ни на какие условия. Вы не можете вызывать организационные блоки, которые вызываются операционной системой в зависимости от события.

Вы можете вызывать функциональные блоки (FB), задав идентификатор блока после оператора CALL, и отделённый запятой идентификатор экземплярного блока данных, связанного с этим вызовом. Вы можете адресовать оба блока или абсолютным способом, или символьным. Назначение абсолютного адреса символьному адресу выполняется в таблице символов (Symbol Table) для экземплярного блока данных, имеющего связанный функциональный блок как тип данных.

При вызове функционального блока нет необходимости инициализировать все параметры вызываемого блока. Неинициализированные параметры блока сохраняют свои текущие значения.

Если Вы создаете функциональные блоки с атрибутом "мультиэкземплярные" ("multi-instance-sarable"), Вы можете также вызывать эти блоки, как локальные экземпляры внутри других "мультиэкземплярных" функциональных блоков. Имеется в виду, что вызванный функциональный блок использует экземплярный блок данных вызывающего функционального блока для сохранения своих собственных локальных данных. В таком случае Вы должны объявить локальный экземпляр в статических локальных данных вызывающего функционального блока, и после этого Вы можете вызывать функциональный блок в программе (без определения экземплярного блока данных). Локальный экземпляр трактуется как сложный тип данных внутри функционального блока "верхнего уровня" (см. раздел 5.5 «Мультиэкземпляры»).

Вы можете вызывать функции (FC), задав идентификатор функции после оператора CALL, или абсолютным способом, или символьным. Далее в инструкции вызова следует список параметров, выводимый редактором в инкрементальном режиме.

При вызове функции Вы должны инициализировать все параметры; тем не менее, параметры могут следовать в любом порядке. Вызываемые функции с функциональным значением имеют точно такую же форму, как и функции без функционального значения. Единственный выходной параметр, соответствующий значению функции, имеет имя RET_VAL.

Операционная система CPU содержит системные функции (SFC) и системные функциональные блоки (SFB), доступные пользователю. Число и тип системных блоков определяется типом CPU. Все системные блоки вызываются оператором CALL. Вы можете вызывать системные функциональные блоки таким же образом, как и те блоки, которые Вы можете написать сами; Вы должны расположить связанный экземплярный блок данных в пользовательской памяти (user memory) с типом данных SFB.

Системные блоки доступны только в операционной системе CPU. При вызове системных блоков во время программирования в автономном (offline) режиме, редактору требуется описание интерфейса вызова для того, чтобы было возможно инициализировать параметры. Описание интерфейса расположено в стандартной библиотеке *Standard library* в системных функциональных блоках *System Function Blocks*. Отсюда редактор копирует описание интерфейса в папку (раздел) автономного режима "Blocks", когда Вы вызываете системный блок. После этого скопированное описание интерфейса вызова появляется как "нормальный" объект блока.

5.3.1. Операторы вызова UC и CC

Вы можете вызывать функциональные блоки и функции с помощью операторов вызова UC и CC. При этом требуется, чтобы вызываемые функции не имели параметров и вызываемые

функциональные блоки не имели экземплярного блока данных и, следовательно, не имели параметров и статических локальных данных. Тем не менее, редактор не проверяет, выполняются ли эти условия.

Для вызова частей блока Вы можете использовать операторы UC и CC, если блок слишком большой и недостаточно удобен для понимания. При этом блок просто разбивается на отдельные фрагменты, после чего эти фрагменты вызываются последовательно. Операторы вызова UC и CC не различают функции FC и функциональные блоки FB. Оба типа блоков обрабатываются одинаковым образом.

Оператор вызова UC является безусловным оператором, что означает, что UC вызывает блок вне зависимости от условий.

Оператор вызова CC является условным оператором, что означает, что CC вызывает блок только при условии, что результат логической операции (RLO) равен "1". Если RLO равен "0", то блок не вызывается, но RLO устанавливается в "1". После этого выполняется следующий за операцией вызова оператор.

Бит состояния OS сбрасывается при смене текущего блока; на биты состояния CC0, CC1 и OV смена блока не влияет, тогда как бит /FC при смене текущего блока сбрасывается.

Вы можете вызывать кодовый блок внутри "двоичного выражения вложения". Текущая глубина стека скобок не изменяется при смене блока. Возможная глубина стека в блоке, который может быть вызван внутри двоичного вложения (binary nest), следовательно, равен разности между максимально возможной глубиной и текущей глубиной вложения при вызове блока.

Содержимое аккумуляторов и адресных регистров не изменяется при вызове блока с помощью операторов вызова UC и CC.

При вызове блока регистр блока данных сохраняется в В-стеке; оператор окончания блока сохраняет его содержимое при завершении обработки вызванного блока.

Блок глобальных данных и экземплярный блок, бывшие текущими перед вызовом блока также открываются после операции вызова блока. Если перед вызовом блока нет открытого блока данных (например, нет блока данных в OB 1), то не будет также открытых блоков данных после операции вызова блока, независимо от того, какие блоки данных открыты в вызванном блоке.

Дополнительные возможности:

- косвенная адресация блоков FB и FC в операторах вызова UC и CC;
- организация вызова с помощью параметров блока в операторе вызова UC;
- организация вызова с помощью параметров блока в операторе вызова CC также и в функциональных блоках.

5.3.2. Операторы окончания блока

Оператор BEC завершает обработку программы в блоке, при этом зависит от состояния RLO, а операторы BEU и BE заканчивают блок независимо от условий.

Выполнение оператора BEC, завершающего обработку программы в блоке, зависит от состояния RLO. Если результат логической операции RLO = "1" при обработке оператора BEC, то функция окончания блока выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что заверщенного блока.

Если RLO = "0" при обработке оператора BEC, то функция окончания блока не выполняется. При этом CPU устанавливает RLO в состояние "1" и выполняется следующая за оператором BEC инструкция, при этом сбрасывается бит /FC. Следующий запрограммированный оператор проверки (опроса) в любом случае является первичным опросом.

Выполнение оператора BEU не зависит ни от каких условий; оператор BEU выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что заверщенного блока.

В отличие от оператора BE оператор BEU может быть использован внутри одного блока несколько раз. При этом отдельные части программы, следующие за оператором BEU, могут быть обработаны только в случае выполнения операции перехода к ним.

Выполнение оператора BE не зависит ни от каких условий: оператор BE выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что заверщенного блока.

Оператор BE всегда является последним оператором в блоке. Использование оператора BE является предметом выбора. При "инкрементном" программировании Вы завершаете программирование блока, закрыв блок.

5.3.3. Временные локальные данные

Временные локальные данные используются для промежуточного хранения результатов, получающихся при обработке программы блока. Временные локальные данные доступны только во время обработки блока; после завершения блока эти данные теряются.

Временные локальные данные соответствуют адресам, которые расположены в стеке локальных данных (в L-стеке) в системной памяти CPU. Операционная система CPU обеспечивает размещение временных локальных данных для каждого кодового блока при вызове этого блока.

Первоначально при вызове блока значения в L-стеке носят "псевдослучайный" характер. Для того, чтобы эти значения отвечали реалиям, до того, как они будут считаны, необходимо их записать. После завершения обработки блока, L-стек назначается следующему вызываемому блоку.

Объем памяти, требуемый для размещения локальных данных для блока, указывается в заголовке блока. Таким образом операционная система узнаёт, сколько байт в L-стеке необходимо отвести для вызываемого блока. Вы также можете узнать из заголовка блока, сколько байт локальных данных требуется для блока (если блок открыт, в редакторе с помощью опций меню: *File -> Properties (Файл -> Свойства)*, или в оболочке SIMATIC Manager при выбранном блоке с помощью опций меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, информация и в том и в другом случае находится на вкладке "General - Part 2" ("Общие - Часть 2").

Объявление временных локальных данных производится в разделе объявлений кодового блока: при инкрементном программировании временные локальные данные помечаются в столбце "Declaration" ("Объявление") словом "temp" (временный).

Временные локальные данные сохраняются в L-стеке в порядке их объявления в соответствии с типами данных.

Адресацию временных локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов, локальных для блока данных).

Все операции, которые действительны для меркеров, разрешены для временных локальных данных. Тем не менее, надо отметить, что биты из временных локальных данных непригодны для использования в качестве меркеров фронта, так как они не сохраняют состояние своего сигнала после окончания обработки блока.

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)
0.0	in	In	INT	0
	out			
	inout			
2.0	stat	Total	INT	0
0.0 2.0	temp temp	temp1 temp2	INT REAL	

Рис. 5.1. Пример объявления локальных данных в функциональном блоке.

Вы можете получить доступ к временным локальным данным блока только непосредственно в самом блоке. (Исключение: к временным локальным данным вызывающего блока можно получить доступ через параметры блока.)

Предельный размер для L-стека определяется типом CPU. Количество байтов временных локальных данных доступное в приоритетном классе, к которому относится программа организационного блока, также фиксировано. Для S7-300 размер L-стека фиксирован: 256 байт на приоритетный класс для CPU 314. При работе с S7-400 размер L-стека может настраиваться пользователем. Необходимое число байт отводится для временных локальных данных при параметризации CPU. Эта область памяти должна быть

общедоступной для блоков, вызов которых производится из соответствующего организационного блока и для блоков, которые вызываются, в свою очередь, из этих блоков.

Необходимо заметить в этой связи, что редактор также использует временные локальные данные, например, при передаче параметров блока. Вы не видите эти временные локальные данные в интерфейсе программирования.

Операционная система CPU передает стартовую информацию (start information) во временных локальных данных при вызове организационного блока. Эта стартовая информация занимает размер памяти 20 байт в каждом организационном блоке и имеет почти идентичную структуру в каждом блоке. Упомянутые 20 байт стартовой информации должны быть всегда доступны в каждом используемом приоритетном классе. Если Вы программируете обработку синхронных ошибок (ошибки программирования и доступа), необходимо обеспечить дополнительные 20 байт, по крайней мере, для стартовой информации этих организационных блоков обработки ошибок, так как эти ОВ ошибок должны обрабатываться в том же приоритетном классе.

Вы должны объявить эту стартовую информацию, при программировании организационного блока. Это обязательное условие. В стандартной библиотеке *Standard Library* имеются шаблоны на английском языке для объявления стартовой информации в организационных блоках. Если Вам не требуется стартовая информация, то достаточно объявить первые 20 байт, например, как массив (как показано на рис. 5.2).

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)
0.0	temp	SINFO	ARRAY [1..20]
*1.0	temp		BYTE
20.0	temp	Lbyte	ARRAY [1..16]
*1.0	temp		BYTE

Рис. 5.2. Пример объявления локальных данных в организационном блоке.

Обычно адресация временных локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Если Вы знакомы с тем, как хранятся данные в L-стеке, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке.

Идентификатором адреса для временных локальных данных является L, при этом биты адресуются идентификатором L, байты - с помощью идентификатора LB, машинные слова - с помощью идентификатора LW, а двойные слова - с помощью идентификатора LD.

Пример: для абсолютной адресации Вы планируете использовать 16 байтов временных локальных данных, к отдельным значениям которых в дальнейшем Вы желаете иметь доступ как в формате байта, так и побитно. Создайте данную область как массив локальных данных. В организационном блоке Вы должны поместить объявление этого массива сразу же за объявлением массива для стартовой информации, таким образом, адресация Вашего массива должна начинаться с адреса 20.

Переменные временных локальных данных могут быть объявлены, как исключение, с типом ANY. При использовании языка STL Вы имеете возможность сгенерировать указатель типа ANY, который может изменяться во время выполнения программы.

5.3.4. Статические локальные данные

Статические локальные данные - это те адреса, которые функциональный блок сохраняет в своем экземплярном блоке данных.

Статические локальные данные - это "память" функционального блока. Эти данные сохраняются до тех пор, пока программа не изменит их, так же как адреса данных в блоках глобальных данных.

Объем памяти, требуемый для размещения статических локальных данных, ограничивается типом данных используемых переменных и максимальным размером блока, определяемым типом используемого CPU.

Объявление статических локальных данных производится в разделе объявлений функционального блока: при инкрементном программировании статические локальные данные помечаются в столбце "Declaration" ("Объявление") словом "stat" (статический).

Статические локальные данные сохраняются в экземплярном блоке после параметров блока в порядке их объявления и в соответствии с типами данных.

Адресацию статических локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов) локальных для блока данных.

Имеется возможность доступа к статическим локальным данным с помощью тех операций, которые действительны для использования в сочетании с адресами в блоках глобальных данных.

Пример. Функциональный блок "Totalizer" (см. рис. 5.3.).

FB "Totalizer" ("Сумматор")				DB "TotalizerData"			
Address адрес	Declaration объявление	Name имя	Type тип	Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT	+0.0	in	In	INT
+2.0	stat	Total	INT	+2.0	stat	Total	INT

Рис. 5.3 Пример статических локальных данных и локальных экземпляров.

Total - это переменная в блоке данных "TotalizerData", который является экземплярным блоком данных для функционального блока "Totalizer" (Вы можете самостоятельно задавать имена всех блоков в таблице символов Symbol Table в соответствии с правилами). Экземплярный блок данных имеет структуру данных функционального блока. В данном примере экземплярный блок данных содержит две переменные формата INT с именами *In* и *Total*.

Статические локальные данные обычно обрабатываются только в функциональном блоке. Тем не менее, хранятся они в блоке данных. Вы можете получить доступ к статическим локальным данным в любое время тем же образом, каким Вы имеете доступ к переменным в блоке глобальных данных с помощью следующей формы адресации: "*ИмяБлокаДанных*". "*ИдентификаторАдреса*".

В нашем маленьком примере блок данных имеет имя "*TotalizerData*", а переменная адресуется как *Total*. Доступ к переменной может иметь следующую форму:

```
L "TotalizerData".Total
T MW 20
L 0
T "TotalizerData".Total
L #In
L #Total
+I
T #Total
```

5.3.5. Функции для блоков данных

Вы можете хранить данные Вашей программы в блоках данных. В принципе, для хранения данных Вы можете также использовать область меркеров (bit memory area). Тем не менее, используя блоки данных, Вы имеете значительно больше возможностей, в плане объема данных, структурирования данных и типов данных.

Существует возможность использовать блоки двух видов: *блоки глобальных данных (global data blocks)*, которые не назначаются никакому кодовому блоку, и *экземплярные блоки (instance data blocks)*, которые назначаются функциональным блокам. Данные блоков глобальных данных являются, попросту говоря,

"свободными" данными, которые может использовать каждый кодовый блок. Вы сами определяете объем, который занимают эти данные, их структуру непосредственно при программировании.

Экземплярные блоки данных содержат только те данные, которые использует связанный функциональный блок; этот функциональный блок определяет структуру данных и место хранения данных в "своем" экземплярном блоке данных.

Число и размер блоков данных определяются типом CPU. Нумерация блоков данных начинается с 1; не может существовать блока данных с именем DB 0. Вы можете использовать каждый блок данных или как блок глобальных данных или как экземплярный блок данных. Вы должны сначала создать блоки данных, которые Вы будете использовать в Вашей программе, или запрограммировав их, так же как кодовые блоки, или используя системную функцию SFC 22 (CREAT_DB) при выполнении программы.

Блоки данных должны быть сохранены в рабочей памяти для того, чтобы к ним был возможен доступ чтения/записи из пользовательской программы. Вы можете также оставить блоки данных в загрузочной памяти, используя при создании блока атрибут "Unlinked". Такие блоки данных не занимают место в рабочей памяти. Тем не менее, Вы сможете только считывать блоки данных из загрузочной памяти с помощью системной функции SFC 20 (BLCMOV) Такая процедура пригодна для блоков данных с данными параметризации или технологическими данными, которые требуются относительно редко для управления установкой или процессом.

Если Вы установили атрибут "The data block is writeprotected in the programmable controller" ("Блок данных защищен от записи в PLC") в свойствах блока, то в дальнейшем Вы сможете только считывать данные из этого DB.

5.3.6. Регистры блоков данных

В CPU есть два "регистра блоков данных" (data block register) для обработки адресованных данных. В этих регистрах содержатся номера обрабатываемых в текущем времени блоков; имеются в виду блоки, в которых находятся обрабатываемые в настоящий момент данные. Перед тем, как получить доступ к адресу, содержащемуся в блоке данных, Вы должны сначала открыть этот блок данных.

Если Вы при адресации данных указываете полный их адрес (о способах спецификации блоков данных см. ниже), то Вам нет необходимости заботиться об открытии блока данных и об использовании регистров блоков данных. Редактор сам создаст необходимые инструкции из Ваших данных.

Редактор использует один из регистров блоков данных преимущественно для доступа к блокам глобальных данных, а второй регистр блоков данных используется для доступа к экземплярным блокам данных. В соответствии с назначением регистров, они имеют следующие идентификаторы: регистр для доступа к блокам глобальных данных: DB-регистр ("Global data block register"), и регистр для доступа к экземплярным блокам данных - DI-регистр ("Instance data block register").

Обработка данных регистров CPU происходит абсолютно одинаково. Каждый блок данных может быть открыт с помощью одного из двух регистров (или же с помощью двух регистров одновременно). При загрузке в аккумулятор слова данных, Вы должны задать, какой из двух возможных открытых блоков содержит это слово данных. Если обрабатываемый блок данных открыт с помощью DB-регистра, то слово данных должно содержать DBW; если обрабатываемый блок данных открыт с помощью DI-регистра, то слово данных должно содержать DIW. Остальные форматы данных также имеют соответствующие обозначения (см. табл. 5.6).

Таблица 5.6 Адресация данных

Адресуемые данные	Блок данных открыт с помощью	
	DB-регистра	DI-регистра
Бит данных	DBX y.x	DIX y.x
Байт данных	DBB y	DIB y
Слово данных	DBW y	DIW y
Двойное слово данных	DBD y	DID y

x = адрес бита, y = адрес байта

5.3.7. Адресация данных

Вы можете использовать следующие способы организации доступа к данным:

- символьная адресация с указанием полного адреса,
- абсолютная адресация с указанием полного адреса,
- абсолютная адресация с указанием неполного адреса (только здесь можно использовать регистр DI).

Символьная адресация с указанием полного адреса глобальных данных в блоке данных требует минимальной информации для системы. Символьная адресация:

- делает более простым чтение и понимание программы (если в качестве символьных имен используются осмысленные обозначения),
- уменьшает ошибки программирования (редактор сравнивает термы, использованные в таблице символов и в программе; "ошибки перестановки чисел", например, DBB 156 и DBB 165, которые могут происходить при использовании абсолютной адресации, не могут произойти при символьной адресации),
- не требует умения программирования на уровне машинных кодов (не требует знания, какой из блоков данных открыт CPU для обработки в настоящий момент).

При символьной адресации указывается полный адрес (указывается и идентификатор блока, и адрес данных), поэтому адрес данных всегда уникален. Символьный адрес данных формируется в два этапа:

- назначение блока данных в таблице символов,
- назначение адресов данных в блоке данных.

Блоки данных являются глобальными данными, имеющими уникальные адреса в программе. В таблице символов Вы должны назначить символьное имя (например, Motor1) абсолютному адресу блока данных (например, DB 51). Вы должны назначить символьные имена адресам данных (и типам данных) во время программирования блока данных. Имена назначаются только в связанном блоке (такие имена являются "внутриблочными"). Вы можете назначать такие же имена другим переменным в другом блоке.

При организации доступа к адресам данных с указанием полного адреса Вы должны задавать адрес данных совместно с адресом блока. Такой способ указания адреса может быть использован как с символьными, так и с абсолютными адресами: MOTOR1.ACTVAL, DB 51.DBW 20. MOTOR1 является символьным адресом, который Вы должны назначить блоку данных в таблице символов. ACTVAL - это адрес данных, который Вы назначаете в блоке данных. Символьное имя MOTOR1.ACTVAL - это такое же уникальное символьное обозначение адреса данных, как и обозначение DB 51.DBW 20.

Доступ к данным посредством полного адреса возможен только при использовании регистра блоков глобальных данных (DB-регистра). При адресации данных с указанием полного адреса редактор выполняет две операции: сначала открывает блок данных с помощью DB-регистра, а затем выполняет операцию доступа к адресам данных.

Вы можете использовать доступ к данным посредством полного адреса при всех разрешенных операциях с адресуемыми типами данных. К таким операциям относятся двоичные логические операции, операции с памятью для адресуемых битов, а также операции загрузки и пересылки для адресуемых численных данных. Вы можете также задавать полный адрес для параметров блоков.

При организации доступа к данным посредством абсолютной адресации Вы должны знать адреса, назначенные редактором для данных, при установке блока. Вы можете найти эти адреса, выведя их после компилирования блока данных. При этом в колонке адресов Вы увидите значения абсолютных адресов, с которых начинаются соответствующие переменные. Такая процедура может быть выполнена для всех блоков данных - тех, которые Вы используете как блоки глобальных данных, и тех, которые Вы используете как экземплярные блоки данных. Таким образом Вы можете видеть, где редактор хранит параметры блоков и статические локальные данные (в случае функциональных блоков).

Организация доступа к адресам данных, имеющих байтовый размер, одинакова, например, с доступом к меркерам; и в том, и в другом случае используются одинаковые операции (см. рис. 5.4), которые выполняются одинаковым способом.

Если Вы намереваетесь назначать исключительно абсолютные адреса данным в блоке данных, Вы должны зарезервировать требуемое количество байтов посредством объявления массива.

5.3.8. Открытие блока данных

Для использования блока данных он может быть открыт с помощью приведённых ниже команд.

OPN DBx - открытие блока данных с помощью DB-регистра с использованием абсолютной адресации;

OPN DBname - открытие блока данных с помощью DB-регистра с использованием символьной адресации;

OPN DIx - открытие блока данных с помощью DI-регистра с использованием абсолютной адресации.

Блоки данных открываются независимо от любых условий. Открытие блоков не действует ни на RLO, ни на содержимое аккумуляторов; глубина вложения вызовов блоков также не изменяется. Открываемый блок должен быть в рабочей (work) памяти.

Пример: значение слова данных DBW 10 из блока данных DB 12 должно быть перенесено в слово данных DBW 12 из блока данных DB 13. Значение слов данных DBW 14 из блоков данных DB 12 и DB 13 должны быть сложены; сумма должна быть сохранена в слове данных DBW 14 из блока данных DB 14. Вы можете запрограммировать данный пример следующими способами организации доступа к данным: с неполной адресацией и с полной адресацией данных

DB 99 "Values":		
Адр.	Имя	Тип
0.0	Status	BOOL
1.0	States	BYTE
2.0	Number	INT
4.0	Weight[1]	REAL
8.0	Weight[2]	REAL

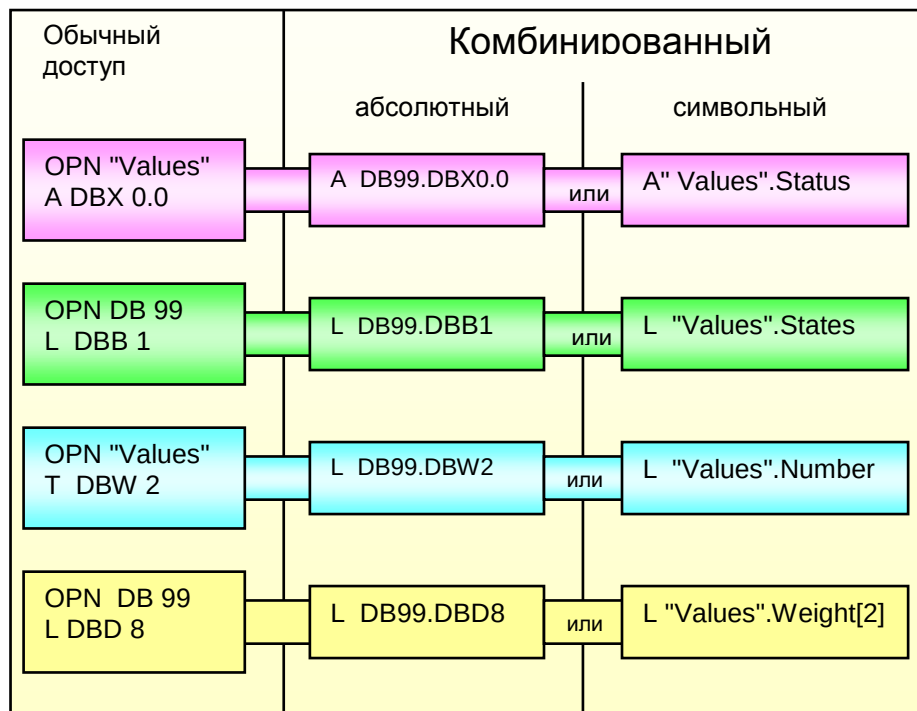


Рис. 5.4. Доступ к элементам данных

```

OPN DB 12
L DBW 10
OPN DB 13
T DBW 12
OPN DB 12
L DBW 14
OPN DB 13
L DBW 14
+I
OPN DB 14
T DBW 14
или
L DB 12.DBW 10
T DB 13.DBW 12
L DB 12.DBW 14
L DB 13.DBW 14
+I
T DB 14.DBW 14

```

Когда блок данных открыт, он остается "доступным" ("valid") до тех пор, пока не будет открыт другой блок данных. При определенных условиях незаметно для пользователя это может обеспечиваться редактором. Например, вызов блока посредством оператора CALL с одновременной передачей параметра может изменять содержимое регистров блоков данных. При смене обрабатываемого блока с помощью операторов UC и CC содержимое регистров блоков данных сохраняется. При возвращении в вызывающий блок оператор завершения блока восстанавливает содержимое регистров.

5.3.9. Размер блока данных и его номер

Для определения содержимого регистров блоков используются следующие операторы:

- L DBLG // Загрузка в аккумулятор 1 размера блока данных, открытого с помощью DB-регистра;
- L DBNO // Загрузка в аккумулятор 1 номера блока данных, открытого с помощью DB-регистра;
- L DILG // Загрузка в аккумулятор 1 размера блока данных, открытого с помощью DI-регистра;
- L DINO // Загрузка в аккумулятор 1 номера блока данных, открытого с помощью DI-регистра.

При выполнении данных операций по внесению номера и размера блока данных в аккумулятор 1, его предыдущее содержимое пересылается в аккумулятор 2, как при "нормальном" выполнении операции загрузки. Если перед операцией загрузки номера или размера блока данных в аккумулятор 1 не было открыто ни одного блока данных, то в результате в аккумулятор будут загружены нулевые (0) значения соответственно в качестве номера или размера блока данных.

Невозможно выполнить запись номера блока в регистр блоков данных прямой операцией загрузки; Вы можете влиять на содержимое регистра блоков данных только посредством операций OPN DB, OPN DI и CDB (обмен содержимым между регистрами блоков данных).

5.3.10. Обмен содержимым между регистрами блоков данных

Оператор CDB позволяет выполнить обмен содержимым между регистрами блоков данных. Операция обмена данными выполняется независимо от условий и не действует ни на биты состояния, ни на другие регистры.

Пример: с помощью оператора CDB Вы можете "обойти" вызов через регистр DB посредством регистра DI (что невозможно выполнить напрямую); номер блока данных пересылается как параметр.

CDB

OPN #Data2

CDB

В примере с помощью оператора CDB содержимое DB-регистра пересылается в DI-регистр. Затем с помощью параметра блока #Data2 открывается блок данных, пересланный как фактический параметр; т.е., номер блока записывается в DB-регистр. После нового обмена содержимого регистров, старое значение вновь возвращается в DB-регистр, а DI-регистр содержит № параметризованного блока данных.

5.3.11. Изменение назначения в DB-регистре

С помощью описанных ниже функций редактор создает дополнительные операторы, которые могут влиять на содержимое одного или двух регистров блоков данных.

Каждый раз, когда при организации доступа к данным используется полная адресация, редактор сначала открывает блок данных с помощью операции OPN DB, затем обеспечивает доступ к адресованным данным. При этом каждый раз производится обновление (переписывание) содержимого DB-регистра. Это же происходит также при инициализации параметров блоков с использованием полной адресации данных.

Содержимое DB-регистра также меняется при операциях доступа к следующим параметрам блоков: для функций, для всех параметров блоков сложных типов и функциональных блоков, входных/выходных параметров сложных типов данных.

Перед тем как выполнить вызов блока, инструкция CALL FB иницирует сохранение номера текущего экземплярного блока данных в DB-регистре (с помощью обмена содержимого в регистрах блоков данных) и открывает экземплярный блок данных для вызванного функционального блока. Таким образом, связанный экземплярный блок данных всегда открыт в вызванном функциональном блоке. Последующие вызовы блока с помощью инструкции CALL FB вновь изменяют содержимое DB-регистра так, что текущий экземплярный блок данных снова доступен для вызванного функционального блока. Таким образом инструкции CALL FB изменяют содержимое DB-регистра.

Для функциональных блоков данных DI-регистр всегда содержит номер соответствующего экземплярного блока данных. Все операции доступа к параметрам блока или к статистическим локальным данным выполняются с помощью DI-регистра и также посредством адресного регистра AR2 в случае "мультиэкземплярных" функциональных блоков.

Примечание: необходимо учитывать указанное выше постоянное назначение DI-регистра, если Вы изменяете содержимое DI-регистра посредством операций CDB или OPN DI.

Если, к примеру, Вы желаете использовать оба регистра блоков данных одновременно для замены данных, то Вы должны сначала сохранить содержимое регистров, чтобы в последствии восстановить

эти значения. Пример, представленный ниже, показывает применение соответствующего приема программирования.

```
ZW_DB WORD //Промежуточный буфер для блока глобальных данных
ZW_DI WORD //Промежуточный буфер для экземплярного блока данных
```

//Сохранение регистров блоков данных

```
L DBNO
```

```
T ZW_DB //Буфер для № блока глобальных данных
```

```
L DINO
```

```
T ZW_DI //Буфер для № экземплярного блока данных
```

// Использование неполной адресации данных и обоих регистров блоков данных

```
OPN DB 12 // Открыть DB 12 с помощью DB-регистра
```

```
OPN DI 13 //Открыть DB 13 с помощью DI-регистра
```

```
L DBW 16 #####
```

```
T DIW 28 ## Будьте осторожны с символьной адресацией
```

```
L DID 30 ## в этой части программы, например, при
```

```
L DBD 30 ## использовании параметров блока,
```

```
+R ; ## "внутриблочных" переменных и при полной адресации данных
```

```
T DID 26 #####
```

//Восстановление регистров блоков данных

```
OPN DB[ZW_DB] //Открыть исходный блок глобальных данных
```

```
OPN DI[ZW_DI] //Открыть исходный экземплярный блок
```

5.3.12. Внесение изменений в назначения блоков данных на поздних стадиях

На вкладке "Blocks" ("Блоки") в окне свойств для папки автономных объектов *Blocks* Вы можете определить, какой из способов адресации данных будет иметь преимущество: абсолютная или символьная - при выполнении изменений в назначениях блока данных для уже сохраненных кодовых блоков.

По умолчанию действует установка "Absolute address has priority" ("Абсолютная адресация имеет приоритет"). Такая, принятая по умолчанию, установка означает, что при изменении в разделе объявлений в блоке, абсолютный адрес сохраняется в программе, а символьный адрес соответственно изменяется. При задании установки "Symbol has priority" ("Символьная адресация имеет приоритет") означает, что при изменении в разделе объявлений в блоке, символьный адрес сохраняется в программе, а абсолютный адрес соответственно изменяется.

Пример: в блоке данных DB 1 слово данных DBW 10 назначено символьному имени *Actual_value*. Если "Data" является символьным именем для блока данных DB 1, то в программе Вы можете загрузить данное слово, например, посредством операции:

```
L "Data".Actual_value DB1.DBW 10
```

Если теперь Вы добавите дополнительное слово данных с помощью символьного имени *MaxCurrent* сразу перед словом данных DBW 10, тогда программа будет иметь следующее содержание при последующем открытии (и сохранении) кодового блока.

Если "Absolute address has priority" ("Абсолютная адресация имеет приоритет"):

```
L "Data".MaxCurrent DB1.DBW 10
```

Если "Symbol has priority" ("Символьная адресация имеет приоритет"):

```
L "Data".Actual_value DB1.DBW 12
```

Такие же условия имеют место при организации доступа к адресам в блоках глобальных данных (например, доступ к адресам входов), если для этих адресов назначены символьные имена в таблице символов.

5.3.13. Системные функции для блоков данных

Существуют три системные функции для обработки блоков данных:

- SFC 22 (CREAT_DB) - с помощью этой функции выполняется создание блока данных,
- SFC 23 (DEL_DB) - с помощью этой функции выполняется удаление блока данных,
- SFC 24 (TEST_DB) - с помощью этой функции выполняется тестирование блока данных.

С помощью системной функции SFC 22 выполняется создание блока данных в рабочей памяти. В качестве номера блока данных системная функция принимает наименьшее свободное число в диапазоне чисел, заданных входными параметрами LOW_LIMIT (нижний предел) и UP_LIMIT (верхний предел). Граничные значения сами входят в диапазон возможных значений для номера блока. Если оба

граничных значения одинаковы, то создаваемый блок будет иметь данное значение. Выходной параметр DB_NUMBER содержит фактический номер созданного блока данных. С помощью входного параметра COUNT Вы задаете размер (длину) создаваемого блока данных.

Длина блока данных соответствует числу байтов данных и при этом должна быть четным числом.

Создание блока данных и вызов блока данных не одно и то же. Текущий (обрабатываемый блок) остается доступным блоком. Блок данных, созданный с помощью системной функции, содержит случайные данные. Для использования блока данных, созданного с помощью системной функции, необходимо сначала определить значения данных, содержащихся в блоке. Только после этого можно будет считывать эти данные в программе.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр DB_NUMBER остается неопределенным, а номер ошибки выдается как значение функции.

С помощью системной функции SFC 23 выполняется удаление блока данных в RAM памяти (рабочей и загрузочной памяти). Номер удаляемого блока данных определен в инструкции во входном параметре DB_NUMBER. При выполнении данной системной функции блок данных не должен быть открыт, иначе CPU перейдет в состояние STOP.

Блок данных, созданный с ключевым словом UNLINKED (несвязанный), а также блок данных, расположенный в модуле памяти FEPR0M не может быть удален с помощью системной функции SFC 23.

Если при выполнении системной функции происходит ошибка, то блок данных не удаляется, а номер ошибки выдается как значение функции.

С помощью системной функции SFC 24 выдается число байтов для блока данных в рабочей (work) памяти в выходном параметре DB_LENGTH, а также ID защиты от записи в выходном параметре WRITE_PROT. Номер тестируемого блока данных Вы задаете в параметре DB_NUMBER.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр остается неопределенным, а номер ошибки выдается как значение функции.

5.4. Параметры блоков

Параметры блока дают возможность передать данные для выполнения инструкций и функций блока.

Пример: необходимо записать блок с функцией сумматора (adder), который Вы хотите использовать в своей программе несколько раз для различных переменных. Переменные будут передаваться как параметры блока; в нашем примере используются три входных параметра и один выходной (см. рис. 5.5). Так как для функционирования сумматора нет необходимости сохранения внутренних значений, то для данной задачи может быть использована функция.

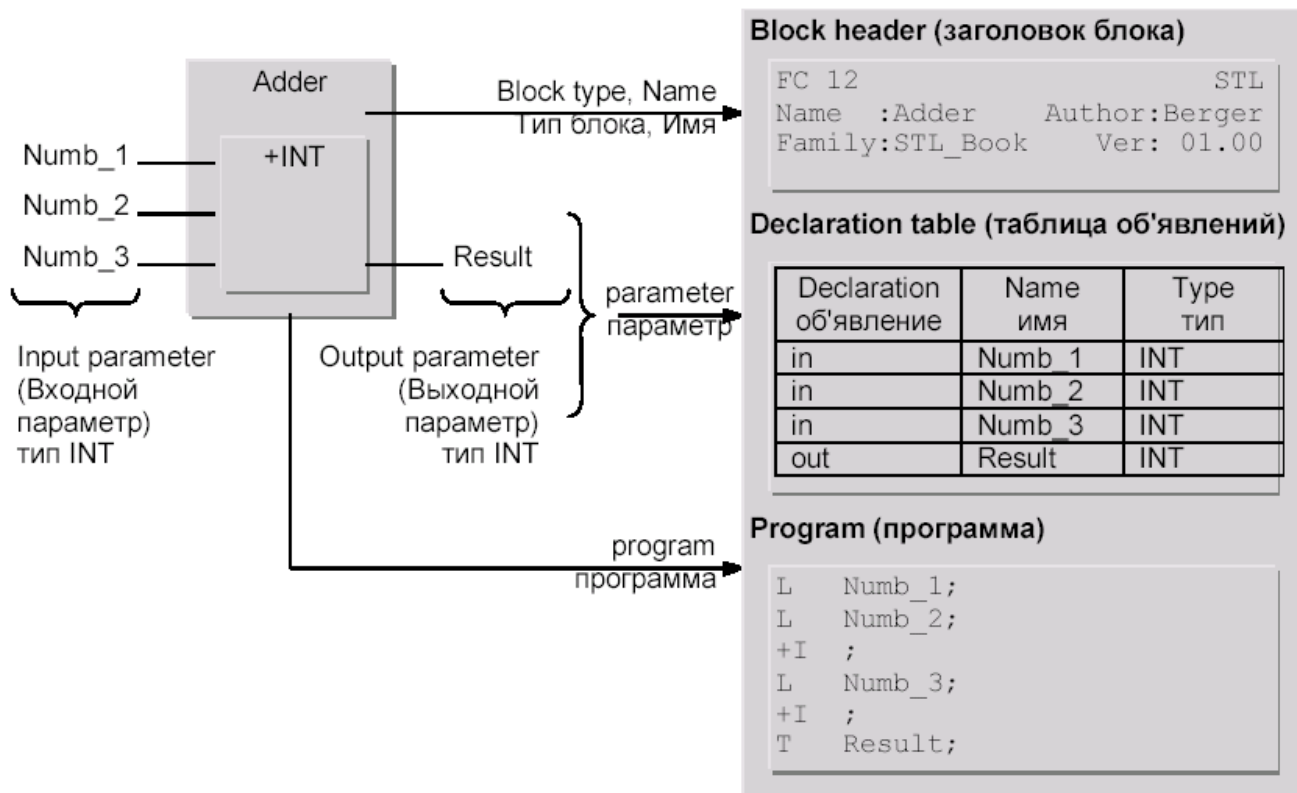


Рис. 5.5. Пример использования параметров блока

Параметр блока необходимо определять как "входной" параметр (input parameter), если требуется только проверить (опросить) или загрузить его значение в программе блока. Если значение параметра только записывается, то Вы имеете дело с "выходным" параметром (output parameter). Если же параметр может быть как записан, так и считан, то Вы должны использовать параметр типа "входной/выходной" (in/out parameter). Редактор не проверяет вариант использования параметров.

5.4.1. Обработка параметров блока

В программе сумматора имена параметров блока являются как бы "вместилищем" для содержимого переменных, которые в дальнейшем будут передаваться в блок для обработки.

Вы можете использовать параметры блока также, как при символьной адресации переменных; в программе они называются *формальными параметрами (Formal parameters)*. Функция "Adder" может быть вызвана в Вашей программе несколько раз. При каждом вызове Вы можете передавать различные значения для операции суммирования с помощью параметров блока (см. рис. 5.6).

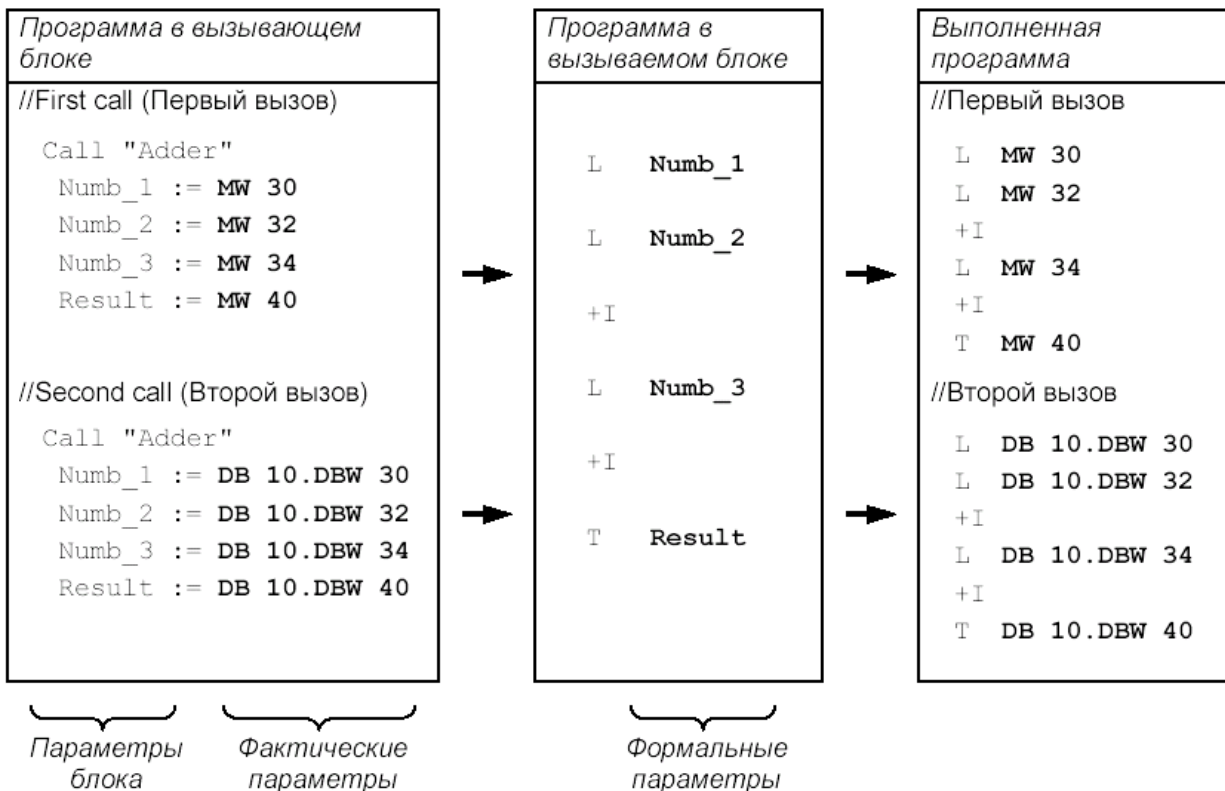


Рис. 5.6 Вызов блока с параметрами

Значения могут быть константами, адресами или переменными; они называются *фактическими параметрами* (*Actual parameters*).

Во время выполнения программы CPU заменяет формальные параметры фактическими параметрами. При первом вызове в примере (рис. 5.6) складываются значения из слов MW 30, MW 32 и MW 34, после чего результат сложения записывается в слово MW 40. Другой блок с другими фактическими параметрами вызывает функцию "Adder" второй раз, в результате чего происходит суммирование данных, заключенных в словах DBW 30, DBW 32 и DBW 34 блока данных DB 10, и сохранение суммарного значения в слове данных DBW 40 блока данных DB 10.

Предопределение параметров является необязательным (опциональным) моментом при программировании и имеет смысл только для функциональных блоков, если значение параметра должно быть сохранено. Это касается всех параметров блоков простых типов, а также входных и выходных параметров сложных типов. Заполнение столбца комментариев для параметров необязательно и может быть выполнено в любой момент.

Имя параметра может содержать до 24 символов. При этом оно должно состоять только из алфавитно-цифровых символов (без национальных элементов, таких как умляут из немецкого алфавита) и символа подчеркивания. Имя не может совпадать ни с одним из ключевых слов. Символы в верхнем и нижнем регистрах не различаются при вводе имени параметра. При выводе программы редактор использует регистр символов, установленный при объявлении имени параметра блока.

При указании типа данных *Data type* для параметров разрешены все типы: простые (*elementary*), сложные (*complex*) и пользовательские (*user-defined*).

STEP 7 сохраняет имена параметров блоков в невыполняемых разделах блоков на носителях данных в программаторе (PG). В рабочей памяти CPU (в скомпилированном блоке) содержатся только объявленные типы (в столбце *Declaration*) и типы данных (в столбце *Type*). Поэтому изменения в программе блока, которые делаются в интерактивном (online) режиме в памяти CPU, должны быть дополнительно продублированы и на носителях данных в PG, чтобы сохранить заданные пользователем имена.

Если не обновить данные в PG или блоки перенести из CPU в программатор, то соответствующие неисполняемые разделы блока будут либо удалены, либо переписаны. Затем редактор перезаписывает имена, заменяя их, для отображения на экране дисплея или для печати (входные параметры при этом получают имена вида INn, выходные параметры получают имена вида OUTn и входные/выходные параметры получают имена вида INOUTn, где n является порядковым номером, начинающимся с 1).

5.4.2. Объявление (declaration) значения функции

Такой параметр блока как значение функции (в случае применения блока-функции) является особым выходным параметром. Он имеет имя RET_VAL (или ret_val) и определяется как первый выходной параметр.

Тип значения функции может относиться к любому из простых типов данных. Кроме того, к допустимым типам в этом случае также относятся следующие типы данных: DATE_AND_TIME, STRING, POINTER, ANY и пользовательский тип UDT. Однако, тип значения функции не может относиться к таким типам данных, как ARRAY и STRUCT.

В рассмотренном выше примере функция "Adder" может быть также запрограммирована с присвоением значения параметра Result значению функции.

При инкрементном программировании Вы должны назначить имя RET_VAL *первому* выходному параметру в списке выходных параметров в разделе объявления переменных. Этим самым Вы определите этот выходной параметр как значение функции для функции FC.

5.4.3. Инициализация (Initialization) параметров блока

При вызове блока Вы инициализируете параметры блока значениями фактических параметров. Эти параметры могут быть константами, абсолютными адресами, данными с полной адресацией или переменными с символьной адресацией. Фактические параметры должны относиться к такому же типу данных, что и параметры блока.

Начиная со STEP 7 V5.1 Вы должны определять параметры блока в исходной программе точно в том порядке, в котором они определены в разделе объявления блока во время программирования. Вы должны инициализировать все параметры функции при каждом ее вызове. В случае использования функциональных блоков инициализация отдельных или всех параметров блока носит необязательный (выборочный) характер.

5.4.4. Формальные параметры

Нет ограничений для доступа к параметрам простых типов, к компонентам полей или структур, к функциям таймеров и счетчиков.

Доступ к параметрам сложных типов и к параметрам типов POINTER и ANY в языке программирования STL в настоящее время не поддерживается. Тем не менее, Вы можете инициализировать соответствующие блоки или системные блоки, которые имеют такие параметры, с помощью соответствующих переменных.

При использовании FC Вы должны передать значение в численный выходной параметр и в значение функции. До этого Вы не должны выходить из функции.

Прямой доступ к параметрам блоков типов ARRAY и STRUCT возможен в "покомпонентном" режиме, т.е., доступ обеспечивается к отдельным двоичным или численным компонентам данных вышеуказанных сложных типов посредством соответствующих операций (двоичных логических операций, операций с памятью, функций загрузки или пересылки).

Доступ к полным переменным (к целым полям или к целым структурам) в настоящее время не поддерживается, а также нет доступа к отдельным компонентам комбинированного или пользовательского типа. Однако, при работе с функциональными блоками Вы можете последовательно передавать ("pass on") параметры типов ARRAY и STRUCT в параметры вызываемых блоков.

В добавление к функциям проверки (опроса), указанным в табл. 5.7, Вы можете программировать обработку параметров блока типа TIMER с помощью следующих операторов:

SP - запуск таймера в режиме "импульса" ("pulse"),
SD - запуск таймера в режиме "задержки включения" ("ON delay"),
SE - запуск таймера в режиме "расширенного импульса" ("extended pulse"),
SS - запуск таймера в режиме "задержки включения с памятью" ("retentive ON delay"),
SF - запуск таймера в режиме "задержки выключения" ("OFF delay"),
R - функция сброса ("reset"),
FR - функция разрешения ("enable"),
используя входной параметр типа TIMER

Таблица 5.7. Доступ к параметрам блока (общий вид)

Тип данных	Допускается для			Доступ к блоку	
	IN	I_O	OUT	возможен	с помощью
Простые типы данных:					
BOOL	x	x	x	x	двоичного опроса, операции с памятью
BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, S5TIME, TIME, DOT, DATE	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Сложные типы данных:					
DT, STRING ARRAY, STRUCT	x	x	x	-	Невозможно в STL непосредственно
Отдельные двоичные компоненты	x	x	x	x	двоичного опроса, операции с памятью
Отдельные двоичные компоненты	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Полные переменные	x	x	x	-	Невозможно в STL непосредственно
Типы параметров:					
TIMER	x	-	-	x	всех операций таймера
COUNTER	x	-	-	x	всех операций счетчика
BLOCK_FC, BLOCK_FB	x	-	-	x	вызова с UC и CC ²⁾
BLOCK_DB	x	-	-	x	открытия с OPN DB
BLOCK_SDB	x	-	-	-	Невозможно ³⁾
POINTER, ANY	x	x	x ¹⁾	-	Невозможно в STL непосредственно

1) Только для функций

2) Оператор CC не применим для функций

3) Имеет значение только для системных блоков

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе "Функции таймера". При запуске функции таймера значение времени "time value" может также быть параметром формата S5TIME.

В добавление к функциям проверки (опроса), указанным в табл. 5.7, Вы можете программировать обработку параметров блока типа COUNTER с помощью следующих операторов:

S - установка счетчика ("set"),

CU - запуск счетчика в режиме "прямого счета" ("count up"),

CD - запуск счетчика в режиме "обратного счета" ("count down"),

R - функция сброса ("reset"),

FR - функция разрешения ("enable"),
используя входной параметр типа COUNTER.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе "Функции счетчика". При запуске функции счетчика значение счетчика "count value" может также быть, например, параметром типа WORD.

Параметры блоков типа BLOCK_xx используются в следующих операторах:

OPN - функция открытия блока данных (параметр типа BLOCK_DB),

UC - вызов функции (параметр типа BLOCK_FC),

UC - вызов функционального блока (параметр типа BLOCK_FB),

CS - условный вызов функции (параметр типа BLOCK_FC),

CS - условный вызов функционального блока (параметр типа BLOCK_FB).

При открытии блока данных посредством параметра блока, CPU всегда использует регистр блоков глобальных данных (DB-регистр). Функции и функциональные блоки, которые пересылаются с параметрами блоков, сами не должны содержать параметров блоков. Условный вызов блока посредством параметра возможен, только если это параметр функционального блока.

Вы можете также использовать блок данных, который переслали как параметр блока, в качестве экземплярного блока. Так как редактор не проверяет тип блока данных, используемого при выполнении программы, Вы должны сами обеспечивать, чтобы пересылаемый блок данных также соответствовал экземплярному блоку данных для вызванного функционального блока.

Пример: Вы можете определить параметр блока типа BLOCK_DB с помощью имени #Data как экземплярный блок данных для вызванного функционального блока: CALL FB 10, #Data

Прямой доступ к параметрам блоков типов POINTER и ANY невозможен. В главе "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам блоков типов POINTER и ANY.

5.4.5. Фактические параметры

При вызове блока Вы инициализируете параметры блока значениями, которые могут быть константами, адресами или переменными, с которыми программа блока должна быть выполнена. Это фактические параметры. При частых вызовах блока в программе, обычно используются различные значения фактических параметров.

Фактический параметр должен относиться к такому же типу данных, что и соответствующий параметр блока. Вы можете поставить в соответствие двоичному фактическому параметру (например, меркеру) только параметр блока типа BOOL; Вы можете инициализировать параметр блока типа ARRAY только переменной, занимающей такое же поле в памяти

При вызове функции Вы должны инициализировать все параметры блока фактическими параметрами.

При вызове функциональных блоков нет необходимости инициализировать параметры блока. STEP 7 сохраняет все параметры блока простых типов, входные и выходные параметры сложных типов и входные параметры типов TIMER, COUNTER и BLOCK_xx как значения или как числа. Вх/вых параметры сложных типов и входные параметры типов POINTER и ANY сохраняются как указатели на фактические параметры. Так как при этом вводится значащее значение, Вы должны инициализировать по крайней мере параметры с неполной адресацией, по крайней мере при первом вызове. Имеется также возможность прямого доступа к параметрам блока. Так как параметры блока располагаются в блоке данных, Вы можете обрабатывать их как адреса данных.

Пример: функциональный блок с экземплярным блоком "Lift_stat_1" управляет двоичным выходным параметром с именем Up. Выполняя обработку программы функционального блока после его вызова, Вы можете проверить (опросить) параметр без инициализации выходного параметра: A "Lift_stat_1".Up Вы можете запрограммировать такую инструкцию проверки (опроса) параметра вместо его инициализации.

Вы можете назначить или абсолютный, или символьный адрес входу, выходу и меркеру. Входные адреса обычно назначаются входным параметрам; выходные адреса обычно назначаются выходным параметрам (тем не менее, это не обязательно). Адреса меркеров допустимы для параметров, объявленных как входные, выходные или вх/вых параметры.

При использовании неполной адресации данных Вы должны обеспечить, чтобы соблюдалась "корректность" в открытом блоке данных при обращении к параметру блока (в вызванном блоке). Так как редактор может в отдельных случаях заменить блок данных при вызове блока, то применение неполной адресации данных не рекомендуется. Поэтому используйте только полную адресацию данных.

Для временных локальных данных обычно используется символьная адресация. Они размещаются в L-стеке вызывающего блока (и объявлены в вызывающем блоке).

Если вызывающий блок является функциональным блоком, Вы можете также использовать его статические локальные данные как фактические параметры.

Для статических данных обычно используется символьная адресация. Если Вы используете абсолютную адресацию с указанием DI-регистра (DI-адресация), то Вы должны обеспечить "корректность" в открытом блоке данных с указанием DI-регистра при обращении к параметру блока (в вызванном блоке).

Примечание: в связи с вышеизложенным надо заметить, что при использовании вызываемых блоков как локальных экземпляров, абсолютные адреса локальных (внутриблочных) переменных зависят от объявления локального экземпляра в вызванном блоке.

В качестве параметра блока типа BOOL Вы можете применять константу TRUE (ИСТИНА - для состояния сигнала "1") или константу FALSE (ЛОЖЬ - для состояния сигнала "0"). В качестве параметра блока численных типов Вы можете применять любые константы, относящиеся к численным типам. Инициализация посредством присвоения констант имеет смысл только для входных параметров.

Вы можете также инициализировать параметры блока простых типов посредством присвоения компонентов полей или структур при условии, что компоненты полей или структур имеют соответствующий тип данных.

Каждый параметр блока может быть сложного типа или пользовательского (UDT) типа. Этим параметрам могут быть поставлены в соответствие адреса переменных соответствующих типов, то есть, эти переменные могут выступать как фактические параметры.

Вы можете для инициализации параметров блока типа DT и STRING использовать отдельные переменные или компоненты полей или структур при условии, что они имеют соответствующий тип данных. Инициализация таких параметров не допустима в STL.

Если Вы инициализируете параметры функционального блока переменной STRING, эта переменная должна иметь такую же длину как и параметр блока STRING.

Когда создается переменная STRING во временных локальных данных, предопределение ее значение невозможно, поэтому при этих условиях содержимое данной переменной содержит "случайные" значения. При использовании такой переменной в качестве фактического параметра для IEC-функции Вы должны сначала определить ее соответствующим корректным значением в программе (перед записью значения в STRING-переменную IEC-функция проверяет его на соответствие типу переменной).

Вы можете для инициализации параметров блока типа ARRAY и STRUCT использовать переменные с точно такой же структурой как и параметры блока.

Для сложных и громоздких структур данных рекомендуется ввести и использовать пользовательский (UDT) тип данных. Сначала Вы должны определить UDT-тип данных, затем - использовать его, например, для создания переменной в блоке данных или для объявления параметра блока. После этого Вы можете использовать переменную при инициализации параметра блока. В этом случае фактический параметр (переменная) должна быть точно такого же типа (такой же UDT-структуры) как и параметр блока.

Вы должны инициализировать параметр блока типа TIMER для функции таймера, параметр блока типа COUNTER для функции счетчика. Для параметров типа BLOCK_FC и BLOCK_FB Вы можете использовать только блоки без своих собственных параметров. После этого эти блоки могут быть вызваны посредством оператора UC (а также CC для функциональных блоков). Инициализировать параметр блока типа BLOCK_DB можно блоком данных, открытым в вызванном блоке посредством DB-регистра.

Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут быть только входными параметрами.

Для параметра блока типа POINTER допустимо использовать только указатели (константы). Эти указатели могут являться указателями либо для области данных (32-разрядные), либо для блока данных DB (48-разрядные). При этом возможна адресация данных простых типов; также возможна полная адресация. Для функциональных блоков не допускаются выходные параметры типа POINTER.

Инициализация параметров блоков типа ANY

Для параметра блока типа ANY допустимо использовать переменные любых типов. При программировании вызываемого блока Вы должны определить, какие переменные (адреса или типы данных) должны быть применены в параметрах блока, какие переменные могут применяться. Вы также можете определить константы в формате указателя ANY ("p#[Data_block.]Address Data_type Number") и таким образом определить область с помощью абсолютной адресации. Исключением является инициализация параметра ANY временными локальными данными типа ANY. В этом случае редактор скорее допустит, что указатель типа ANY уже существует во временных локальных данных, чем создаст указатель на переменную. Это дает Вам возможность применить для параметра ANY указатель ANY, которым можно манипулировать во время выполнения программы. "Переменный указатель ANY" может быть полезен при работе с системной функцией SFC 20 (BLKMOV).

Для функциональных блоков не допускаются выходные параметры типа ANY.

5.4.6. Последовательная передача ("Pass On") параметров блока

Последовательная передача ("Pass On") параметров блока - это особая форма доступа и инициализации параметров блока. При такой форме доступа параметры блока вызывающего передаются ("passed on") в параметры вызванного блока. При этом формальный параметр вызывающего блока становится фактическим параметром вызванного блока.

В общем случае, при этом фактический параметр должен быть того же типа, что и формальный параметр (то есть, параметры блока должны соответствовать типам передаваемых данных). Вдобавок, Вы можете применить входной параметр вызывающего блока только как входной параметр вызываемого блока, и, аналогично, использовать выходные параметры. Вы можете применить вх/вых параметр вызывающего блока во всех (входных, выходных и вх/вых) параметрах вызываемого блока.

Существуют ограничения, зависящие от типов данных, которые возникают из-за различий в способе хранения параметров блоков между функциями и функциональными блоками. Параметры блоков могут последовательно передаваться ("pass on") без ограничений и в соответствии с информацией, изложенной в предыдущих параграфах. Сложные типы данных для входов и выходных параметров могут передаваться, если только вызывающий блок является функциональным блоком. Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут передаваться от одного входного параметра к другому, только если вызывающий блок является функциональным блоком. В табл. 5.8 представлены разрешенные и запрещенные пути передачи данных между входными и выходными параметрами для функций и функциональных блоков в зависимости от типа передаваемых данных.

Таблица 5.8. Разрешенные комбинации для последовательной передачи ("pass on") параметров

Вызывающий ----> вызываемый (объявленный тип)	FC вызывает FC			FB вызывает FC			FC вызывает FB			FB вызывает FB		
	E	C	P	E	C	P	E	C	P	E	C	P
Input -> Input (Bx -> Bx)	x	-	-	x	x	-	x	-	x	x	x	x
Output -> Output (Вых -> Вых)	x	-	-	x	x	-	x	-	-	x	x	-
In/Out -> Input (Bx/Вых -> Bx)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> Output (Bx/Вых -> Вых)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> In/Out (Bx/Вых -> Bx/Вых)	x	-	-	x	-	-	x	-	-	x	-	-

E = простые типы данных

C = сложные типы данных

P = параметрические типы TIMER, COUNTER и BLOCK_xx

Вы можете последовательно передавать ("pass on") данные типов TIMER, COUNTER и BLOCK_xx при работе с функциями при использовании косвенной адресации. Соответствующему параметру сначала должен быть присвоен тип данных WORD или INT; затем Вы инициализируете его константой или переменной, которая содержит номер таймера, счетчика или передаваемого блока. Вы можете последовательно передавать ("pass on") этот параметр, так как он относится к параметрам простого типа. В последнем (в цепочке передачи) блоке Вы можете использовать функцию загрузки "load" для передачи содержимого параметра в слово временных локальных данных, после чего может быть выполнена функция таймера, счетчика или блока.

5.5. Мультиэкземпляры

В дополнение к экземплярам функциональных блоков, когда экземпляр DB определяется в запросе FB, STEP 7 также поддерживает явную декларацию экземпляров FB в пределах функционального блока более высокого уровня (см. рис. 5.7). Для этого экземпляры вызываемых функциональных блоков объявляются в

разделе описаний вызывающего функционального блока FB 10, как статические переменные с типами данных FB1 или FB2, используя символьные идентификаторы (Drill, Motor_1 и Motor_2). В пределах функционального блока более высокого уровня отдельные экземпляры вызываются, используя их символьные идентификаторы.

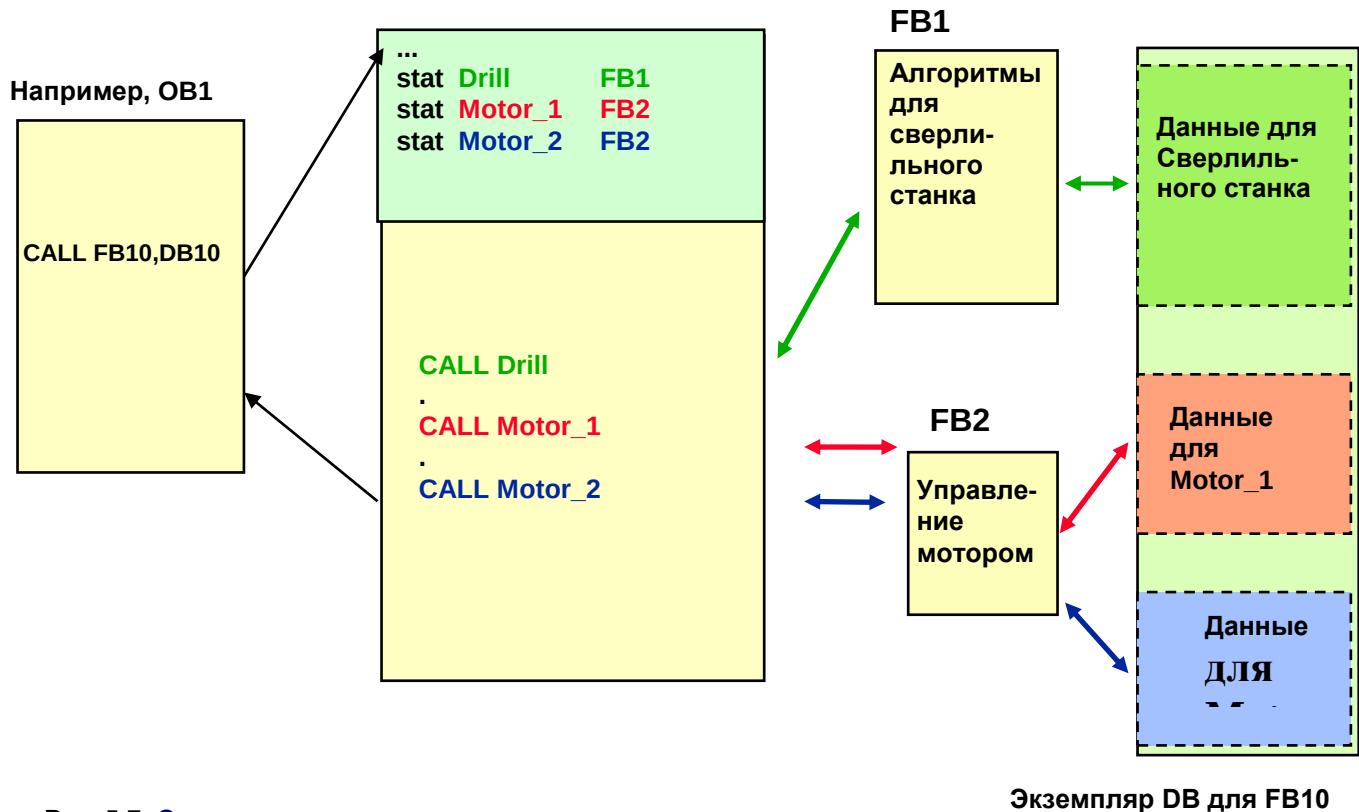


Рис. 5.7. Структура модели мультиэкземпляров

Функциональный блок FB10 более высокого уровня должен, однако, вызываться с собственным экземпляром DB (DB10). STEP 7 проверяет в созданном экземпляре DB более высокого уровня наличие области данных, требуемых для отдельных вызываемых FB-экземпляров.

В вызове отдельных экземпляров используются символьные имена, макрокоманда CALL устанавливает в регистр AR2 указатель на начало области данных, назначенной экземпляру.

Использование модели мультиэкземпляров имеют следующие преимущества.

- Отдельные экземпляры не требуют собственного блока данных. В пределах иерархии вызовов функциональных блоков только один экземпляр DB "потрачен впустую" при вызове функционального блока, вызывающего все остальные.
- Модель мультиэкземпляров "сваривает" функциональный блок и область данных экземпляра в один объект (экземпляр FB), который может быть обработан как одна единица. Программист не должен заботиться об управлении (создание, адресация) отдельными областями данных экземпляра. Он должен просто обеспечить экземпляр DB для "главного" FB.
- Модель мультиэкземпляров поддерживает объектно-ориентированный стиль программирования.

5.5.1. Локальные экземпляры

При вызове функционального блока обычно назначается также экземплярный блок данных. Функциональный блок сохраняет свои параметры и свои статические локальные данные в этом экземплярном блоке.

Начиная с STEP 7 V2, Вы можете создавать "мультиэкземпляры", что означает, что Вы можете вызывать одни функциональные блоки в других функциональных блоках. Статические локальные данные (и параметры блока) вызванного функционального блока являются подмножеством статических локальных данных вызывающего блока. Для этого требуется, чтобы вызываемый и вызывающий функциональные блоки имели версию 2, чтобы они могли быть обработаны в режиме "мультиэкземпляра". Таким путем можно организовывать вложение вызовов функциональных блоков с глубиной вложения до 8 вызовов.

FB "Totalizer" ("Сумматор")

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

DB "TotalizerData"

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

FB "Evaluation"

Address адрес	Decla- ration объяв- ление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat	Memory	Totalizer

DB "EvaluationData"

Address адрес	Decla- ration объяв- ление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat	Memory.In	INT
6.0	stat	Memory. total	INT

Рис. 5.8. Пример статических локальных данных и локальных экземпляров.

Пример (рис. 5.8 (внизу)): в статических локальных данных функционального блока "Evaluation" объявлена переменная *Memory*, которая соответствует функциональному блоку "Totalizer" и имеет такую же структуру. Теперь Вы можете вызывать функциональный блок "Totalizer" посредством переменной *Memory*, без определения блока данных, так как данные для *Memory* размещены внутриблочно ("block-local") в статических локальных данных (*Memory* является "локальным экземпляром" блока "Totalizer"):

```
A #Add
FP #EM_Add
JCN M1
CALL #Memory
In := "Value2"
M1: A #Delete
FP #EM_Del
JCN End
L #Memory.Total
T "Result"
L 0
T #Memory.Total
```

Доступ к статическим локальным данным *Memory* в программе функционального блока "Evaluation" выполняется таким же способом, как и к компонентам структуры с определением имени структуры (*Memory*) и имени компонента (*Total*).

Экземплярный блок данных "EvaluationData", следовательно, содержит переменные *Memory.In* и *Memory.Total*, к которым Вы можете также обращаться как к глобальным переменным, например, следующим образом: "EvaluationData".*Memory.Total*.

Чтобы использовать FB как мультиэкземпляр необходимо твердо придерживаться следующих пунктов.

- При управлении производственным процессом недопустим никакой прямой доступ к глобальным адресам CPU (типа входов и выходов). Каждый доступ к глобальному входу и выходу нарушает возможность многократного использования.
- Связь с процессом или с другими секциями программы (FB) должна быть выполнена, используя только параметры FB. Только после того как FB интегрирован в единицу более высокого уровня, можно сделать "назначения" через список параметров FB в его вызове.
- Состояния или другая информация относительно объекта, которыми нужно управлять, должны "запоминаться" FB в собственных статических переменных.

- Вначале создать FB, используемый как мультитекст, затем вызывающий его FB, в котором описать мультитексты как статические переменные FB-типа. После этого организовать вызов вызывающего мультитекста с указанием блока данных. Загрузить объекты в порядке их создания.

5.5.2. Абсолютная адресация статических локальных данных

Обычно адресация статических локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Внутри функционального блока экземплярный блок данных открывается посредством регистра DI. Идентификатором для адресов в блоке данных, для статических локальных данных также как параметров блока является идентификатор DI. Биты адресуются идентификатором DIX, байты - с помощью идентификатора DIB, слова - с помощью идентификатора DIW, а двойные слова - с помощью идентификатора DID.

Если Вы знакомы с тем, как хранятся данные в блоке данных, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке. Но будьте внимательны! *Эти адреса связаны с запуском экземпляра.* Они действительны только в случае, если Вы вызываете функциональный блок с блоком данных. Если Вы вызываете функциональный блок как локальный экземпляр, соответствующие локальные данные этого локального экземпляра располагаются внутри экземплярного блока данных вызывающего функционального блока. Вы можете просмотреть абсолютные адреса, например, в скомпилированном экземплярном блоке данных, в котором содержатся все локальные экземпляры. Для просмотра адресов отдельных локальных данных выберите опции меню: *Select View -> Data View (Выбор просмотра -> Просмотр данных).*

Вернемся к нашему примеру. В функциональном блоке "Totalizer" переменная *Total* может быть адресована с помощью DIW 2, если FB "Totalizer" вызывается с блоком данных (см. назначение адресов в DB "TotalizerData"), и с помощью DIW 6, если FB "Totalizer" вызывается как локальный экземпляр в блоке FB "Evaluation" (см. назначение адресов в DB "EvaluationData"). Тем не менее, при программировании функционального блока, если пока неизвестно, будет ли он вызываться с блоком данных или будет вызываться как локальный экземпляр, то возникает вопрос: как в этом случае назначать абсолютные адреса статическим локальным данным? В этом случае к адресу переменной прибавляется смещение локального экземпляра из адресного регистра AR2.

5.6. Выполнение программы

Основная программа (main program) выполняется циклически. После каждого прохода программы CPU возвращается к началу программы и вновь запускает ее на выполнение. Это "стандартный" метод выполнения программ для PLC.

Многочисленные системные функции поддерживают использование системных служб, таких, как управление системными часами или связь посредством системной шины. В отличие от статических установок, выполненных при параметризации CPU, системные функции могут использоваться в динамическом режиме в процессе выполнения программы.

Основная программа может быть на время приостановлена для того, чтобы выполнить **обслуживание прерываний** (interrupt servicing). Различные типы прерываний: аппаратные прерывания (hardware interrupts), таймерные прерывания (watchdog interrupts), временные (по времени суток) прерывания (time-of-day interrupts), прерывания с задержкой обработки (time-delay interrupts), прерывания мультипроцессорного режима (multiprocessor interrupts) - разбиты по приоритетным классам. Приоритеты обработки Вы можете в большей степени определять самостоятельно.

Обработка прерываний позволяет пользователю оперативно реагировать на сигналы, поступающие от управляемого процесса, а также позволяет периодически выполнять процедуры контроля (мониторинга и управления) вне зависимости от времени обработки основной программы.

Перед запуском основной программы на выполнение CPU активизирует **программу запуска** (start-up program), при помощи которой Вы можете определить параметры, относящиеся к выполнению основной программы, определить значения переменных, принимаемые по умолчанию, а также Вы можете параметризовать модули.

Обработка ошибок (error handling) является также необходимой частью обработки программы. В STEP 7 различаются синхронные ошибки (ошибки, возникающие во время обработки инструкций) и асинхронные ошибки (ошибки, которые случаются вне зависимости от выполнения программы). Для тех и других ошибок Вы можете создать программу обработки ошибок (error routine) в соответствии с Вашими требованиями.

Основная программа (main program) - это циклически выполняемая (иначе говоря, "сканируемая") программа пользователя. "Циклическое сканирование" - это обычный метод выполнения программ в программируемых логических контроллерах. Подавляющее большинство систем управления использует только такой способ выполнения программ. Если применяется сканирование программы, управляемой событиями, то в большинстве случаев такая программа лишь дополняет основную программу.

Основная программа вызывается в организационном блоке ОВ 1. Она имеет самый низкий приоритет, и ее выполнение может быть прервано любыми другими процедурами обработки программы. При выполнении программы переключатель режимов на передней панели CPU должен быть в положении RUN или RUN-P. Если переключатель режимов находится в положении RUN-P, то возможно программирование посредством программатора. Если переключатель режимов находится в положении RUN, то Вы можете удалить ключ, так что никто не сможет изменить рабочий режим без подтверждения прав авторизации. Если переключатель режимов находится в положении RUN, то программа защищена от изменений - возможно только ее считывание.

5.6.1. Структура программы

Проанализировать сложную задачу автоматизации - это значит разделить эту задачу на ряд мелких задач или функций в соответствии со структурой процесса, для которого необходимо построить систему управления. Затем необходимо спланировать те отдельные функциональные части, которые получены при разбиении общей задачи автоматизации процесса, чтобы определить заложенные в них функции и сигналы связи (интерфейса) с процессом и другими функциональными частями. Это разделение общей задачи управления на отдельные функциональные части может быть выполнено в Вашей программе. Таким образом, структура Вашей программы должна соответствовать структуре задачи автоматизации. Структурированность программы пользователя может сделать программу более легко конфигурируемой. Такая программа может быть написана отдельными блоками (в том числе и несколькими программистами, в случае большого объема программы). И, в конце концов, не менее важный момент - структурированная пользовательская программа всегда является более простой в пуско-наладочных работах и обслуживании. Структурированность программы зависит от ее размера и ее функций.

Различаются три варианта структуры программы:

- линейная программа,
- фрагментированная программа,
- структурированная программа.

Линейная программа характерна тем, что основная программа целиком располагается в организационном блоке ОВ 1. Программа разбивается на сегменты (networks). STEP 7 последовательно нумерует сегменты. При редактировании или отладке программы Вы можете ссылаться на сегмент непосредственно по его номеру.

Фрагментированная программа характерна тем, что, по существу оставаясь линейной, программа разбивается на отдельные блоки. Причиной такого разбиения программы могут быть или слишком большой размер программы для размещения ее в организационном блоке ОВ 1, или необходимость улучшения читаемости программы. Блоки программы вызываются последовательно. Программа в отдельном блоке может быть таким же образом разбита, как и в организационном блоке ОВ 1. Такой способ программирования позволяет вызывать отдельные программы, связанные с отдельными функциями процесса, из разных блоков. Преимущество такой структуры программы заключается также в том, что несмотря на то, что программа остается линейной, ее можно отлаживать и выполнять отдельными фрагментами (просто добавляя или пропуская отдельные вызовы тех или иных блоков).

Структурированная программа используется, если концептуальная формулировка задачи управления особенно широка, если необходимо повторно использовать функции программы, если требуется решать сложные задачи. Структурирование программы означает разбиение программы на части (блоки), которые включают в себя независимые функции или которые имеют особое функциональное назначение и при этом обмениваются минимально возможным количеством сигналов с другими блоками.

Назначение для каждого блока программы особой функции (связанной с процессом) делает программируемые блоки легко читаемыми и с более простым интерфейсом с другими блоками программы.

Язык программирования STL поддерживает структурное программирование посредством функций, с помощью которых Вы можете создавать блоки (автономные части программы). Блоки также формируют некоторые результаты обработки данных; и эти результаты также должны быть возвращены посредством того же интерфейса для последующей обработки.

Способ организации программы определяет, будет ли и в каком порядке будет CPU обрабатывать блоки программы, которые Вы создали. Чтобы организовать соответствующим образом Вашу программу,

необходимо запрограммировать вызовы блоков в требуемой последовательности в вызывающих блоках. Необходимо также определить порядок, в котором блоки должны вызываться, и который отображает функциональную организацию установки (процесса), для которой требуется построить систему управления.

Глубина вложения (Nesting depth) вызовов - это параметр, который зависит от приоритетного класса (касается программ в организационном блоке), а также зависит от типа CPU. В CPU 314, например, глубина вложения имеет значение восемь, что означает, что, начиная с организационного блока (1-й уровень вложения), Вы можете добавить еще 7 блоков "по горизонтали" (это называется вложением). Если будет последовательно (блок из блока) вызвано более 7 блоков, тогда CPU перейдет в режим STOP с индикацией ошибки переполнения стека блоков ("Block overflow"). При подсчете глубины вложения не забывайте учитывать вызовы системных функциональных блоков (SFB), а также вызовы системных функций (SFC).

Вызовы блоков данных, которые фактически открывают или выбирают области данных, не влияют на глубину вложения блоков, как не влияет на глубину вложения последовательный вызов нескольких блоков "по вертикали" (последовательный вызов блоков, скажем, в организационном блоке OB 1).

В организационном блоке OB 1 Вы должны таким образом выстроить последовательность вызовов блоков в основной программе, чтобы в первом приближении выполнялась логическая организация Вашей программы. При этом, в основе своей, программа может быть организована или с ориентацией на процесс ("process-related"), или с ориентацией на выполнение функций ("function-related").

Последующие моменты обсуждения этого вопроса могут дать только приблизительное, очень общее представление с целью дать начинающему пользователю некоторые идеи относительно структурирования программы и относительно перенесения его задачи управления в практическую плоскость. Продвинутые программисты обычно имеют уже достаточный опыт для того, чтобы сразу организовать структуру программы, отвечающую требованиям задачи управления.

Структура программы с ориентацией на процесс ("process-related") близка к структуре установки, для которой требуется система управления. Отдельные части программы соответствуют отдельным частям установки или управляемого процесса. В общих чертах такая структура предполагает и сканирование устройств блокировки, и использование панелей операторов, и управление приводами, и устройствами отображения (в различных частях установки). При этом для обмена сигналами между различными частями установки используются меркеры или глобальные данные.

Структура программы с ориентацией на выполнение функций ("function-related") базируется на обеспечении функций управления. Изначально (в общей схеме) этот метод структурного программирования вообще не берет в расчет установку, для которой проектируется система управления. Структура установки начинает обнаруживать себя лишь в программных блоках более высокого (следующего) уровня разбиения (при дальнейшем разбиении структуры программы).

На практике обычно используется смешанный, гибридный подход, содержащий оба вышеуказанных метода построения логической схемы программы для системы управления. Функциональная структура отражается в программе режимов работы ("operating mode program") и в программе обработки данных ("data processing program"), которые "сверху" и "снизу" ограничивают блоки, относящиеся собственно к установке.

5.6.2. Управление циклом сканирования

Область памяти для отображения состояния процесса находится во внутренней системной памяти CPU. Область памяти для отображения состояния процесса начинается с адреса 0 и заканчивается некоторым верхним предельным значением адреса, который зависит от типа CPU. Отдельные типы CPU позволяют пользователю самостоятельно определять верхнее значение адреса области отображения состояния процесса.

Обычно все дискретные модули отображаются в адресной области отображения состояния процесса, в то время как аналоговые модули отображаются в адресной области за пределами области отображения состояния процесса. Если CPU позволяет свободное размещение адресов, Вы можете использовать таблицу конфигурации (configuration table) для размещения любого модуля в адресной области отображения состояния процесса или за пределами области отображения состояния процесса.

Образ процесса состоит из таблицы отображения входов процесса (inputs, I) и из таблицы отображения выходов процесса (outputs, Q). После перезапуска CPU перед первым выполнением организационного блока OB 1 операционная система пересылает состояния сигналов таблицы отображения выходов процесса в выходные модули и считывает сигналы входных модулей в таблицу отображения входов процесса. За этим следует выполнение организационного блока OB 1; в процессе которого обычно происходит обработка входных сигналов от входов процесса (inputs I) и вырабатываются сигналы

управления (выходные сигналы) для выходов процесса. После этого следует завершение обработки блока OB 1 и обновление образа процесса перед началом нового цикла выполнения организационного блока OB 1 (см. рис. . 5.9.).

Если случается ошибка во время обновления образа процесса, например, если недоступен какой-либо модуль, то вызывается организационный блок обработки ошибок, возникающих при выполнении программы, OB 85 ("Program Execution Errors"). Если блок OB 85 недоступен, то CPU переходит в состояние STOP.

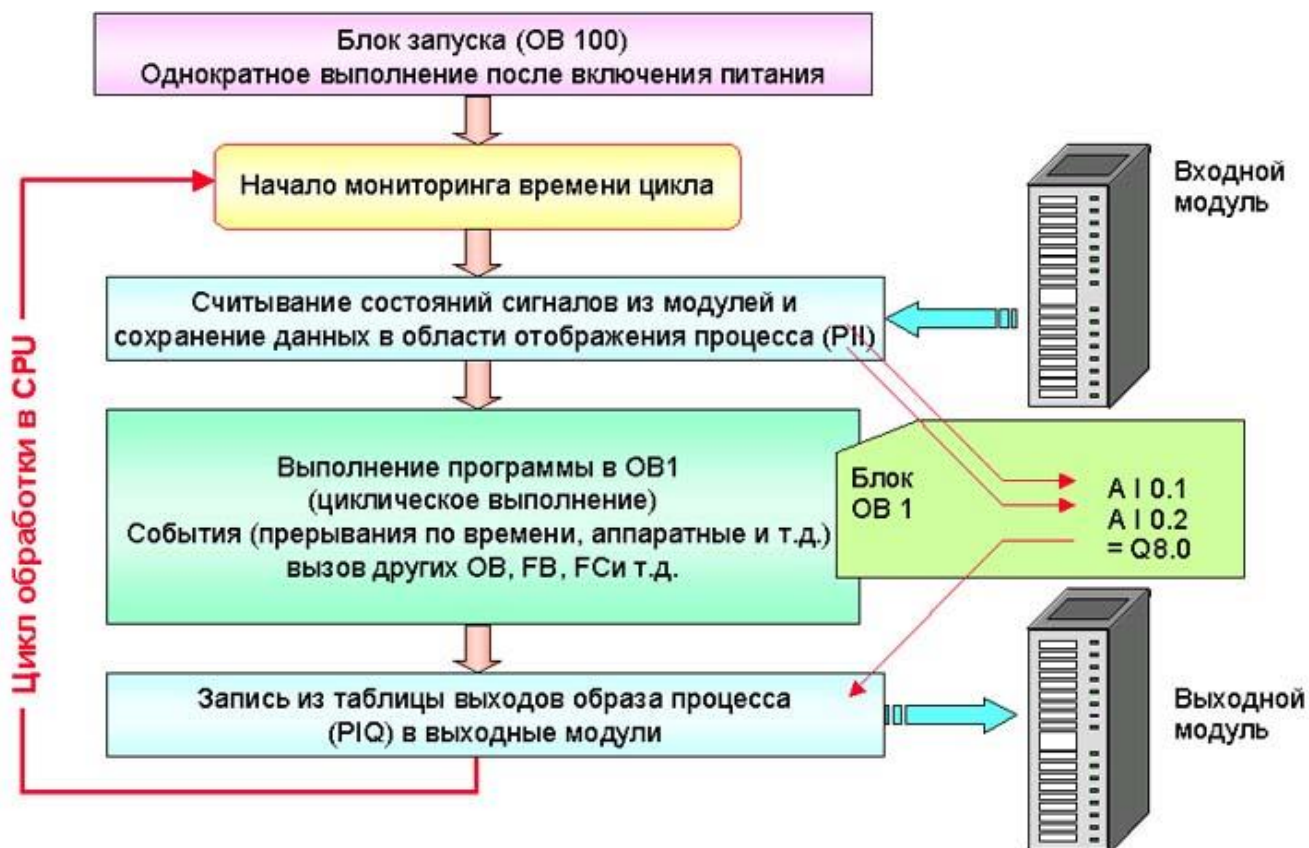


Рис. 5.9. Циклическое выполнение программы

Для отдельных, специально предназначенных для этого CPU, Вы можете разбивать область отображения состояния процесса на отдельные подобласти (для отображения состояния подпроцессов) общим числом от 9 до 16. Такое разбиение области отображения состояния процесса на отдельные подобласти выполняется во время параметризации сигнальных модулей, в результате чего модули должны получить адреса.

Вы можете выполнить разбиение области отображения в соответствии с разбиением на таблицы отображения входов и таблицы отображения выходов. Все модули, для которых не будет сделано назначение в одном из адресных полей для образов подпроцессов с номерами 1 ... 8 или 1 ... 15, будут адресоваться в адресном поле для образа подпроцесса с номером 0. Этот образ подпроцесса 0 обновляется автоматически операционной системой CPU в процессе циклического выполнения программы.

Для отдельных, специально предназначенных для этого CPU, Вы можете также назначать области отображения состояния для организационных блоков обработки прерываний, которые автоматически обновляются при вызове этих OB.

Для обновления образа подпроцесса используются системные функции, которые вызываются из программы пользователя. Для обновления таблицы отображения входов подпроцессов используется системная функция SFC 26 (UPDAT_PI), а для обновления таблицы отображения выходов подпроцессов используется системная функция SFC 27 (UPDAT_PO). С помощью этих функций Вы можете также обновлять образы подпроцессов 0. Вы можете выполнить обновление образа отдельного подпроцесса, вызывая указанные системные функции в любое время и в любом месте. Например, Вы можете определить образ подпроцесса для приоритетного класса (для уровня выполнения программы), затем Вы можете вызывать обновление данного образа подпроцесса в начале и в конце выполнения соответствующего организационного блока при обработке этого приоритетного класса.

Обновление образа процесса может быть прервано вызовом более высокого приоритетного класса. Если случается ошибка во время обновления образа процесса, например, если более недоступен модуль, то SFC возвращает код ошибки.

Сканирование программы в организационном блоке OB 1 отслеживается с помощью так называемого "монитора цикла сканирования" ("scan cycle monitor") или "таймера цикла сканирования" ("scan cycle watchdog").

Значение, которое принимается по умолчанию для времени мониторинга цикла сканирования, равно 150 мс. Вы можете изменять это значение в пределах от 1 мс до 6 с путем соответствующей параметризации CPU. Если сканирование основной программы выполняется за больший промежуток времени, чем установленное время мониторинга цикла сканирования, тогда CPU вызывает OB 80 ("Timeout" - "Превышение времени"). Если организационный блок OB 80 не запрограммирован, то CPU переходит в состояние STOP.

Время мониторинга цикла сканирования соответствует полному времени сканирования для OB 1. В это время также входит время сканирования блоков с более высоким приоритетным классом, которые вызываются в основной программе (в текущем цикле). Процессы связи, выполняемые операционной системой, например, GD-коммуникации или операции доступа программатора к CPU, также увеличивают время выполнения основной программы. Время выполнения основной программы может быть уменьшено в некоторой степени при параметризации CPU (на вкладке "Cycle/Clock memory bits" ["Цикл/Тактовые меркеры"] - "Cyclic load from communication" ["Циклическая загрузка посредством коммуникаций"]).

В случае наличия интерактивной (online) связи программатора с CPU с помощью выбора опций: *PLC -> Module Information (PLC -> Информация о модуле)*, Вы можете вызвать диалоговое окно с несколькими вкладками.

На вкладке "Cycle Time" ("Время цикла") будет показано текущее значение длительности времени цикла, а также длительности самого короткого и самого продолжительного циклов сканирования. На этой же вкладке отображаются также минимальная длительность цикла и время мониторинга цикла сканирования ("scan cycle monitoring time"). Кроме того, продолжительность последнего цикла сканирования также как и продолжительности минимального и максимального по длительности циклов с момента последнего запуска PLC, могут быть считаны из временных локальных данных в стартовой информации организационного блока OB 1.

Вызов системной функции SFC 43 (RE_TRIGR) перезапускает время мониторинга цикла сканирования; таймер перезапускается с новым значением, установленным посредством параметризации CPU. Функция SFC 43 не имеет параметров.

Время мониторинга цикла сканирования также включает в себя продолжительность процедур операционной системы. К этим процедурам относятся:

- системное управление циклическим сканированием ("no-load cycle" - "незагружаемый цикл"); фиксированное значение;
- обновление образа процесса; зависит от числа байт, которые необходимо обновлять;
- обновление таймеров; зависит от числа таймеров, которые необходимо обновлять;
- загрузка коммуникаций.

Функции связи (коммуникации) для CPU включают в себя функции передачи блоков программы пользователя и обмен данными между модулями CPU с использованием системных функций. Время, которое CPU будет затрачивать на выполнение этих функций, может быть ограничено при параметризации CPU.

Все значения продолжительностей процедур операционной системы зависят от типа CPU и являются его характеристиками.

Для отдельных, специальным образом оборудованных, CPU Вы можете задавать минимальное время цикла сканирования ("minimum scan cycle time"). В случае если продолжительность выполнения основной программы (включая обработку прерываний) меньше минимального времени цикла сканирования, CPU будет ожидать, пока не истечет заданное минимальное время, и только после этого начнется следующий цикл с повторного вызова OB 1.

Значение, принимаемое по умолчанию для минимального времени цикла сканирования, составляет 0 мс, иначе говоря, данная функция неактивна (disabled). Для параметра "минимальное время цикла сканирования" ("minimum scan cycle time") Вы можете установить значение в пределах от 1 миллисекунды до 6 секунд на вкладке "Cycle/Clock memory bits" ("Цикл/Тактовые меркеры") при параметризации CPU.

В интервале между фактическим окончанием цикла сканирования и моментом истечения времени минимальной продолжительности цикла сканирования CPU обрабатывает организационный блок OB 90 "Background scanning" ("Сканирование в фоновом режиме"). Организационный блок OB 90 выполняется "по частям". Когда операционная система вызывает блок OB 1, выполнение блока OB 90 прерывается; при этом, когда обработка блока OB 1 закончится, выполнение блока OB 90 продолжится в точке прерывания. Выполнение организационного блока OB 90 может быть прервано после каждого оператора; тем не менее,

любой системный блок, вызванный в блоке ОВ 90, перед обработкой прерывания будет выполнен полностью. Размер отдельных выполняемых фрагментов блока ОВ 90 зависит от продолжительности текущего цикла сканирования блока ОВ 1. Чем меньше разница между продолжительностью текущего цикла сканирования и минимальной продолжительностью цикла сканирования, тем меньше времени остается для выполнения ОВ 90.

Мониторинг времени сканирования программы в ОВ 90 не проводится. Блок ОВ 90 выполняется (сканируется) только в режиме RUN. Этот процесс сканирования может быть прерван, если возникает событие прерывания или ошибки, так же как прерывается обработка блока ОВ 1.

5.6.3. Время отклика ("Response Time")

Если программа пользователя в ОВ 1 работает со значениями (с состояниями сигналов) из области отображения процесса, это сказывается на таком параметре, как время отклика ("response time"), который зависит от времени выполнения программы (от времени сканирования цикла). Величина времени отклика имеет значение, лежащее между длительностью одного и двух циклов сканирования, это показано в следующем примере (см. рис. 5.10)..

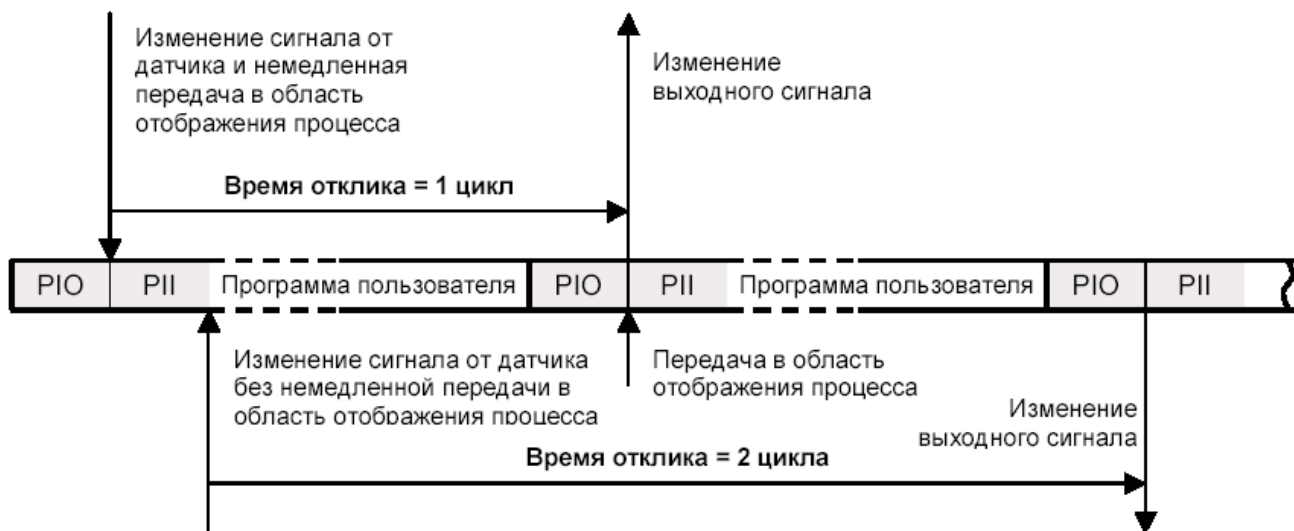


Рис. 5.10. Время отклика программируемого контроллера

Если, например, "концевой выключатель" ("limit switch") активирован, то состояние его сигнала изменяется со значения "0" на значение "1". Программируемый контроллер обнаруживает это изменение сигнала во время последующего обновления образа процесса и устанавливает входы, соответствующие этому концевому выключателю в состояние "1". В программе происходит проверка состояния сигнала от концевой выключателя, и при обнаружении состояния "1" происходит сброс некоторого выхода для выключения соответствующего двигателя. Новое состояние сигнала этого выхода, который был сброшен, передается в конце цикла сканирования программы; только после этого соответствующий бит в выходном дискретном модуле будет сброшен.

Наилучший случай имеет место, когда образ процесса обновляется сразу же после изменения состояния сигнала, поступающего от концевой выключателя. При этом проходит период времени, равный только одному циклу сканирования, от момента изменения сигнала выключателя до момента изменения состояния соответствующего выхода.

Наихудший случай имеет место, когда изменение состояния сигнала, поступающего от концевой выключателя, происходит сразу же после того, как произошло обновление образа процесса. При этом проходит период времени, приблизительно равный одному циклу сканирования, до момента, когда PLC обнаруживает изменение сигнала от выключателя и устанавливает соответствующий ему вход. После этого должен пройти еще один период времени, равный одному циклу сканирования, прежде чем будет изменено состояние соответствующего выхода.

Время, необходимое для выполнения программы пользователя, включает в себя время выполнения всех процедур выполняемых в одном программном цикле (включая, например, обслуживание прерываний, выполнение операционной системой таких функций, как обновление таймеров, управление MPI-интерфейсом, обновление образа процесса).

Величина времени отклика на изменение входного сигнала, таким образом, может иметь значение, лежащее между длительностью одного и двух циклов сканирования. Величина времени отклика PLC увеличивается также за счет времени задержки во входных модулях, времени переключения контакторов и т.п.

В некоторых случаях Вы можете уменьшить время отклика посредством прямой адресации I/O периферии или с помощью организации вызовов разделов программы на основе управления событиями.

5.6.4. Стартовая информация

Операционная система CPU первоначально обращается к стартовой информации в организационном блоке OB1, как и при обработке любого другого организационного блока. Стартовая информация ("start information") занимает первые 20 байт во временных локальных данных. Вы можете самостоятельно создать объявление для стартовой информации или использовать для этого информацию из стандартной библиотеки *Standard Library* в разделе для организационных блоков. В таблице 5.9 представлены стартовая информация для организационного блока OB1, назначение символьных имен, принимаемых по умолчанию, и типы данных. Вы можете изменять назначения в любое время и можете, также, выбрать более приемлемые для Вас имена. Даже если Вы не используете стартовую информацию, Вы должны зарезервировать первые 20 байтов во временных локальных данных для стартовой информации (например, в форме массива размером 20 байтов).

В SIMATIC S7 все сообщения о событиях имеют фиксированную структуру, которая определяется классом события. В стартовой информации блока OB 1, например, сообщение о событии - V#16#11 - стандартный вызов OB. Из содержания следующего байта Вы можете узнать, находится ли основная программа в первом цикле после включения, а также были ли после этого вызваны, например, программы инициализации в циклической программе. Приоритет и номер OB основной программы имеют фиксированные значения. В стартовой информации содержатся три числа в формате INT, показывающие величину предыдущего цикла сканирования, а также величины максимального и минимального циклов с момента последнего включения контроллера. Последнее значение в формате DATE_AND_TIME показывает, когда программа управления обнаружила событие для вызова организационного блока OB 1.

Таблица 5.9. Стартовая информация для организационного блока OB 1

Имя	Тип данных	Описание	Содержание
OB1_EV_CLASS	BYTE	Класс события	V#16#11
OB1_SCAN_1	BYTE	Стартовая информация	V#16#01 = 1-й цикл после полного перезапуска V#16#02 = 1-й цикл после теплого перезапуска V#16#02 = каждый другой цикл
OB1_PRIORITY	BYTE	Приоритет	V#16#01
OB1_OB_NUMBR	BYTE	Номер OB	V#16#01
OB1_RESERVED_1	BYTE	Зарезервировано	
OB1_RESERVED_2	BYTE	Зарезервировано	
OB1_PREV_CYCLE	INT	Время предыдущего цикла в мс	
OB1_MIN_CYCLE	INT	Время минимального цикла в мс	
OB1_MAX_CYCLE	INT	Время максимального цикла в мс	
OB1_DATE_TIME	DT	Время прихода события для вызова OB 1. Время вызова OB (в цикле)	

Надо отметить, что прямое считывание стартовой информации организационного блока возможно выполнять только в этом организационном блоке, так как эта информация содержится во временных локальных данных. Если Вам требуется стартовая информация, которая находится в более "глубоких" уровнях вложения вызовов, то Вы должны использовать вызов системной функции SFC 6 (RD_SINFO) в соответствующем месте программы.

Системная функция SFC 6 обеспечивает считывание стартовой информации в текущем организационном блоке (то есть, в OB, находящемся на вершине "дерева вызовов") и в последнем выполненном организационном блоке запуска даже на более глубоком уровне вызовов.

Системная функция SFC 6 может быть вызвана не только в любом месте основной программы, но и в каждом приоритетном классе, даже в организационном блоке обработки ошибок или в программе запуска.

5.6.5. Управление часами реального времени (Real-Time Clock)

Следующие системные функции могут использоваться для управления часами реального времени CPU (Real-Time Clock):

- SFC 0 (SET_CLK) - Функция позволяет установить дату и время,
- SFC 1 (READ_CLK) - Функция позволяет считать дату и время,
- SFC 48 (SNC_RTCB) - Функция позволяет синхронизировать часы CPU.

Если несколько CPU связаны между собой посредством подсети, "часы" одного из них должны быть инициализированы как "master clock" ("главные часы"). При параметризации CPU также необходимо задать интервал синхронизации, по прошествии которого все часы в подсети должны автоматически синхронизироваться с "master clock" ("главными часами").

При вызове функции SFC 48 в CPU с назначенными "master clock" ("главными часами") происходит синхронизация всех часов в подсети, независимо от автоматической синхронизации. При установке времени в "master clock" (в главных часах) посредством системной функции SFC 0 происходит автоматическая синхронизация всех часов в подсети - все часы будут установлены на заданное время.

Системные часы CPU (system clock) запускаются при включении питания или при полном перезапуске. Системные часы CPU идут (включены) пока CPU выполняет программу перезапуска или находится в режиме RUN. Когда CPU переходит в режим STOP или HOLD, текущее "системное время" (system time) "замораживается".

Если Вы активируете "теплый" ("warm") перезапуск в системе с S7-400 CPU, системные часы вновь начинают отсчет с ранее запомненного "замороженного" значения времени.

При "холодном" перезапуске или полном перезапуске происходит сброс системного времени. Системное время имеет формат данных TIME, поэтому его величина может принимать только положительные значения в диапазоне от TIME#0ms до TIME#24d20h31m23s647ms.

В случае переполнения значения системного времени, оно вновь начинает отсчитываться с 0. В CPU 3xx (кроме CPU 318) обновление системного времени происходит каждые 10 миллисекунд, а в CPU 4xx, а также в CPU 318 обновление системного времени происходит каждую миллисекунду.

Системная функция SFC 64 (TIME_TCK) обеспечивает считывание текущего системного времени. Её параметр RET_VAL содержит значение системного времени в формате TIME. Вы можете использовать системные часы, например, для считывания текущего времени выполнения программы CPU или, вычисляя разность их показаний в двух точках, рассчитать величину промежутка времени между двумя вызовами функции SFC 64. Разность между двумя значениями в TIME-формате вычисляется командами для DINT-чисел.

Измеритель времени наработки (run-time meter) в CPU отсчитывает наработанные аппаратурой часы. Вы можете определять наработанное время CPU или подключенных к нему устройств. Возможное число измерителей наработки для каждого CPU определяется его типом. Если CPU переходит в режим STOP или HOLD, измеритель времени наработки также останавливается; когда CPU вновь запускается, измеритель времени наработки продолжает отсчет времени с запомненного в момент остановки значения.

Когда показания измерителя времени наработки достигают значения 32767 часов, измеритель времени наработки останавливается и сигнализирует о переполнении значения. Измеритель времени наработки может быть установлен на новое значение или сброшен в нулевое состояние только при вызове соответствующей системной функции SFC.

Следующие системные функции используются для управления измерителем времени наработки :

- SFC 2 (SET_RTM) - функция позволяет установить измеритель времени наработки,
- SFC 3 (CTRL_RTM) - функция позволяет остановить или запустить измеритель времени наработки,
- SFC 4 (READ_RTM) - функция позволяет считать показания измерителя времени наработки.

5.6.6. Сжатие информации в памяти CPU (Compressing CPU Memory)

Многочисленное удаление и повторная загрузка блоков (что часто происходит во время интерактивного изменения блоков) могут приводить к появлению незанятых участков в рабочей памяти CPU и в загрузочной RAM-памяти, что в свою очередь, уменьшает количество полезного пространства памяти. Для устранения таких незанятых участков в памяти служит функция "Compress" ("Сжатие"), при запуске которой происходит заполнение незанятых участков путем сдвига блоков друг к другу. Вы можете

инициировать функцию "Compress" ("Сжатие") с помощью программатора PG, подключенного к CPU, или с помощью вызова системной функции SFC 25 (COMPRESS).

Системная функция SFC 25 не может быть активизирована, если запущен процесс сжатия извне, если активна функция удаления блоков "Delete Block" или если функции из PG получили доступ к блоку, который должен быть сдвинут (например, функция статуса блока "Status Block").

Необходимо отметить, что отдельные блоки максимальной длины (определяется типом CPU) не могут быть сдвинуты, т.е. незанятые участки в памяти CPU остаются. Только функция "Compress" ("Сжатие"), запущенная из программатора, в то время, пока CPU находится в режиме STOP, может полностью устранить незанятые участки в памяти CPU.

5.6.7. Мультипроцессорный режим

Станция S7-400 может использоваться в мультипроцессорном режиме. В одной стойке при этом могут работать до четырех допускающих такой режим процессоров. Станция S7-400 автоматически запускается в мультипроцессорном режиме после того, как в центральной стойке Вы установили посредством утилиты конфигурирования аппаратной части Hardware Configuration больше, чем один CPU. При этом слоты для установки CPU произвольны.

CPU различаются по автоматическим назначаемым номерам, идущим в порядке возрастания в соответствии с местом (слотом) установки. Пользователь также может назначить номер CPU самостоятельно используя вкладку "Multicomputing" ("Мультипроцессорный режим").

Данные конфигурирования для всех CPU должны быть загружены в PLC, даже если Вы делаете изменения в конфигурации только одного CPU. После назначения параметров для CPU Вы должны назначить каждый модуль станции соответствующему CPU. Это делается с помощью параметризации модуля на вкладке "Address" ("Адреса") в окне "CPU assignment" ("Назначение CPU").

Одновременно с назначением адресной области модуля Вы должны назначить прерывания модуля для этого CPU. С помощью опций: *View -> Filter -> CPU No.x-Modules (Bild -> Фильтр -> CPU No.x)* Вы можете выделить модули, назначенные для CPU в таблицах конфигурации.

Все CPU в мультипроцессорной сети имеют одинаковый режим работы. Это означает, что:

- все они должны быть параметризованы с одинаковым режимом перезапуска;
- все они должны переходить в режим RUN одновременно;
- все они должны быть переведены в режим HOLD, когда Вы делаете отладку на одном из CPU;
- все они должны переходить в режим STOP, как только один из CPU переходит в режим STOP.

Как только происходит сбой в одной стойке в станции, в каждом CPU должен вызываться организационный блок OB 86. Программы пользователя в таких CPU выполняются независимо одна от другой; они не синхронизированы.

Вызов системной функции SFC 35 (MP_ALM) запускает организационный блок OB 60 "Multiprocessor interrupt" ("Обработка прерывания мультипроцессорного режима ") на всех CPU одновременно.

5.7. Коммуникации посредством глобальных данных

Коммуникации посредством глобальных данных (GD-коммуникации) - это особая система связи (коммуникационная служба), встроенная в операционную систему CPU, которая используется для обмена небольшими объемами некритичных ко времени данных с использованием MPI-шины. К данным, которые могут переноситься с помощью глобальных данных, относятся:

- данные входов и выходов (отображения процесса);
- данные меркеров;
- данные блоков DB;
- данные таймеров и счетчиков как данные, которые необходимо переслать.

Для использования GD-коммуникаций необходимо обеспечение ряда требований: все CPU должны быть включены в сеть посредством MPI-интерфейса или подключены к К-шине как в монтажной стойке S7-400, все CPU должны существовать в одном проекте STEP 7, чтобы можно было сконфигурировать GD-коммуникации.

Для коммуникаций посредством глобальных данных не обязательно использовать операционную систему: существуют системные функции для GD-коммуникаций в S7-400.

Необходимо отметить, что CPU-приемник не подтверждает факт приема глобальных данных. Следовательно, CPU-передатчик не получает никакого ответного сообщения, из которого следовало бы, что "приемник" получил данные, и если получил, то какие именно. Тем не менее, Вы имеете возможность мониторинга состояния связи между двумя CPU, так же как и состояния всех GD-групп CPU.

Посылка и прием глобальных данных зависят от параметра, известного как "частота сканирования" ("scan rate"). Этот параметр определяется количеством циклов сканирования пользовательской

программы, после выполнения которого CPU осуществляет передачу или прием данных. Каждый раз процессы передачи данных и их приема между "передатчиком" и "приемником" происходят синхронно в определенной точке цикла, то есть в некоторый момент по окончании сканирования циклически выполняемой программы и перед новым циклом ее выполнения (как, например, при обновлении образа процесса).

Обмен данными происходит в форме пересылки пакетов данных (GD packet) между CPU, сгруппированными в GD-группы (GD circle).

CPU, которые обмениваются общими (shared) GD-пакетами, формируют GD-группы. GD-группы могут быть следующих видов.

- Одностороннее соединение CPU, который посылает GD-пакеты данных в адрес нескольких других CPU, которые принимают эти пакеты данных.
- Двустороннее соединение между двумя CPU, при котором каждый из этих двух CPU может посылать GD-пакет данных другому.
- Двустороннее соединение между тремя CPU, при котором каждый из этих трех CPU может послать один GD-пакет данных двум другим CPU (только для S7-400 CPU).

В одной GD-группе могут обмениваться данными друг с другом до 15 единиц CPU. Один CPU может при этом принадлежать к нескольким GD-группам.

GD-пакет включает в себя заголовок пакета из восьми байтов и один или несколько элементов глобальных данных (GD-элементов):

- ID 1-го GD-элемента (2 байта),
- пользовательские данные 1-го GD-элемента (x байтов),
- ID 2-го GD-элемента (2 байта),
- пользовательские данные 2-го GD-элемента (x байтов)
- и т.д.

Каждый GD-элемент состоит из 2 байтов описания и данных, передаваемых по сети. 3 байта требуются для передачи байта данных (байта меркеров), 4 байта требуются для передачи слова данных (слова меркеров), 6 байт требуются для передачи двойного слова данных (двойного слова меркеров). Булева переменная занимает 1 байт данных, передаваемых по сети, следовательно, она занимает такой же объем как и переменная длиной один байт. Значения таймеров и счетчиков, имеющие размер 2 байта каждый, в GD-пакете занимают по 4 байта.

GD-элемент может также включать в себя данные из адресной области. Например, запись MB 0:15 означает область байтов от MB 0 до MB 15, а DB20.DBW14:8 представляет область данных, размещенных в блоке DB20, которая начинается со слова данных DBW14 и содержит 8 слов данных.

Максимальный размер GD-пакета составляет 32 байта для S7-300 и 64 байта для S7-400.

Максимальное число байтов данных, передаваемых по сети посредством GD-пакета, составляет до 22 байтов для S7-300 и до 54 байтов для S7-400.

Консистентность данных обеспечивается для одного GD-элемента. Если GD-элемент больше, чем объем данных, для которого гарантируется консистентность, то консистентные данные соответствующего размера располагаются, начиная с первого байта.

5.7.1. Конфигурирование GD-коммуникаций

Для того, чтобы использовать возможности GD-коммуникаций у Вас должен быть создан проект, в котором должна быть доступна подсеть MPI и должны быть сконфигурированы S7-станции. В каждой станции должен быть по крайней мере один CPU. Используя кнопку "Properties" ("Свойства") в MPI-интерфейсе на вкладке "General" ("Общие") окна свойств CPU (можно открыть, дважды щелкнув на строке CPU в окне конфигурации оборудования или в строке, содержащей submodule MPI-интерфейса), Вы можете установить MPI-адрес и выбрать подсеть MPI для подключаемого CPU.

Конфигурирование GD-коммуникаций производится путем заполнения **таблицы глобальных данных**. Вы можете вызвать пустую таблицу посредством выбора значка подсети MPI в Simatic Manager или при конфигурировании сети (Network Configuration) с помощью опций меню: *Options -> Define Global Data* (Опции -> Определить глобальные данные). Выберите в таблице столбец, а затем опции меню: *Edit -> CPU* (Правка -> CPU). В левой половине открывшегося окна для выбора проекта выберите станцию, затем в правой половине окна выберите CPU. Выбор CPU для таблицы глобальных данных подтверждается нажатием клавиши "OK".

Выполните те же действия в отношении остальных CPU, участвующих в обмене посредством глобальных данных. Таблица глобальных данных может содержать до 15 столбцов CPU.

Для конфигурирования обмена данными между CPU посредством GD-коммуникаций выберите первую строку под CPU-передатчиком и задайте адрес, значение которого должно передаваться (завершить действие клавишей Enter).

С помощью опций меню: *Edit -> Sender (Правка -> Передающий CPU)* Вы определяете значение, которое должно быть передано, обозначаемое символом-префиксом ">" и выделяемое тенью. В той же строке под CPU-приемником Вы определяете адрес, в который должно быть принято данное значение (свойство "Receiver" ["Приемник"] устанавливается по умолчанию). Функции таймера и счетчика Вы можете использовать только как "передатчики"; "приемником" данных и от функции таймера, и от функции счетчика должен служить адрес области, имеющей размер слова данных.

Строка может содержать несколько "приемников", но при этом может иметь только один "передатчик" (см. таблицу 5.10). После заполнения такой таблицы Вы должны ее скомпилировать. Для этого выберите опции меню: *GD Table -> Compile (GD-таблица -> Компилировать)*.

Таблица 5.10. Пример GD-таблицы с указанием состояния (GST) и частоты сканирования (SR)

GD-идентификатор	CPU 417 (3)	CPU 414 (4)	CPU 416 (5)	CPU 315 (7)	CPU 314 (10)
GST	MD100	MD100	MD100	DB10.DBD200	DB10.DBD200
GDS 1.1	DB9.DBD0		MD92	DB10.DBD204	DB10.DBD204
SR 1.1	44	0	44	8	8
GD 1.1.1	>DB9.DBW10		MW90	DB10.DBW208	DB10.DBW208
GDS 2.1	MD96	MD96			
SR 2.1	44	23	0	0	0
GD 2.1.1	>MW10:10	DB3.DBW20:10			

После компиляции (фаза 1) созданных системных данных достаточно для обеспечения обмена посредством глобальных данных. Если Вы также конфигурируете состояние GD-коммуникаций и частоту сканирования Вы снова должны выполнить компилирование GD-таблицы.

После выполнения без ошибок процесса компилирования GD-таблицы STEP 7 заполняет столбец "GD ID". Этот столбец показывает пользователю структуру передаваемых данных в GD-группах, GD-пакетах и GD-элементах.

После выполнения процесса компилирования GD-таблицы Вы можете задать адреса для формирования состояния (статуса) GD-коммуникаций. Это выполняется с помощью опций меню: *View -> GD Status (Вид -> состояние GD-коммуникаций)*. Параметр общего состояния (статуса) (GST) определяет все коммуникационные соединения в таблице.

Параметр состояния (статуса) (GDS) показывает состояние коммуникационных соединений для передачи GD-пакетов. В каждом случае параметр использует двойное слово данных.

Осуществление функций связи посредством глобальных данных требует значительной доли времени на выполнение соответствующих операций в общем объеме работы операционной системы CPU, а также требует определенных затрат времени на передачу по MPI-шине. Для того, чтобы свести такие затраты времени ("коммуникационная нагрузка") к минимуму, можно соответствующим образом настраивать параметр "Scan rate" ("Частота сканирования"). Этот параметр определяет количество циклов сканирования пользовательской программы, после выполнения которых CPU осуществляет передачу или прием данных в форме GD-пакета.

Так как с учетом применения параметра "Scan rate" обновление данных происходит не в каждом цикле сканирования программы, пользователю рекомендуется избегать использования GD-коммуникаций для пересылки данных, критичных ко времени.

После первого выполнения процесса компилирования GD-таблицы (если не произошло ошибок) Вы можете самостоятельно определить параметры "Scan rate" ("SR") для каждого GD-пакета и для каждого CPU. Это выполняется с помощью опций меню: *View -> Scan Rates (Вид -> Частота сканирования)*. Стандартно параметр "Scan rate" устанавливается таким, что при "пустом" (без пользовательской программы) CPU посылка и прием GD-пакета осуществляется приблизительно каждые 10 миллисекунд. Если после этого загрузить в CPU пользовательскую программу, интервал между соседними посылками или приемами GD-пакета возрастает. Пользователь может задавать параметр "Scan rate" в диапазоне от 1 до 255.

Необходимо отметить, что при уменьшении значения "Scan rate" сканирования) "коммуникационная нагрузка" CPU возрастает.

Для того, чтобы поддерживать коммуникационную нагрузку CPU в приемлемых рамках, необходимо устанавливать такие значения параметра "Scan rate" для CPU-передатчика, чтобы произведение этого значения на величину времени сканирования для S7-300 было больше 60 мс, а для S7-400 было больше 10 мс.

Соответственно для CPU-приемника необходимо устанавливать такие частоты сканирования, чтобы произведение их значений на величину времени сканирования для S7-300 было меньше 60 мс, а для S7-400 было меньше 10 мс. Выполнение указанных рекомендаций позволяет избежать потерь отдельных GD-пакетов.

Установив нулевое ("0") значение параметра "Scan rate", с использованием SFC-функций Вы можете включить обмен данными посредством соответствующего GD-пакета, обусловленный только событиями.

Функции связи посредством глобальных данных становятся доступными после того, как GD-таблица пересылается в подключенные CPU с помощью опций меню: *PLC -> Download (PLC -> Загрузить)*.

GD-коммуникации становятся доступными также после того, как будет переслан объект *System data* (системные данные), который содержит все установки для оборудования и установки для параметров.

В системах S7-400 Вы можете из программы пользователя управлять обменом данными посредством GD-коммуникаций. В дополнение к циклической передаче глобальных данных или вместо циклической передачи глобальных данных Вы можете организовать передачу или прием GD-пакета посредством следующих системных функций:

- SFC 60 (GD_SND) - функция пересылки GD-пакета,
- SFC 61 (GD_RCV) - функция приема GD-пакета.

Необходимым условием для использования этих системных функций является наличие сконфигурированной таблицы глобальных данных. После компилирования этой таблицы STEP 7 в столбце "GD Identifier" (Идентификатор GD-коммуникаций) показывает пользователю структуру передаваемых данных - номера GD-групп и GD-пакетов, которые необходимо учитывать при назначении параметров функций.

Функция пересылки GD-пакета SFC 60 вводит GD-пакет в системную память CPU и инициализирует его пересылку; функция приема GD-пакета SFC 61 позволяет выбрать GD-пакет из системной памяти CPU. Если параметр "Scan rate" имеет ненулевое значение в GD-таблице, то циклическая передача данного GD-пакета данных также имеет место.

Если необходимо обеспечивать консистентность данных для GD-пакета в целом при пересылке с использованием системных функций SFC 60 и SFC 61, то пользователь должен заблокировать (disable) или отложить на время (delay) выполнение прерываний с высоким приоритетом, как и обработку асинхронных ошибок, как на "передающей" (Send) так и на "принимающей" (Receive) стороне, во время выполнения функций SFC 60 и SFC 61.

Системные функции SFC 60 и SFC 61 не обязательно вызывать парами; и "смешанные" операции также возможны. Например, Вы можете использовать функцию SFC 60 (GD_SND) для обеспечения пересылки GD-пакетов, обусловленной событиями с последующим циклическим их считыванием.

5.8. Обработка прерываний

Обработка прерываний всегда обусловлена событиями. Когда операционная система получает сигнал о том, что произошло некоторое событие, она прекращает сканирование основной программы и вызывает определенную программу, соответствующую этому конкретному событию. После выполнения этой программы операционная система продолжает сканирование основной программы с точки прерывания. Такие прерывания обработки программы могут иметь место после каждой операции (оператора).

Упомянутыми событиями могут быть как собственно "прерывания" (interrupts) так и ошибки. Порядок, в котором обрабатываются фактически одновременно происходящие прерывания, регулируется планом приоритетов. Каждое событие имеет свой приоритет обработки. Отдельные прерывания могут быть объединены в классы приоритетов (priority class - "приоритетный" класс).

Каждая программа, связанная с событием-прерыванием, должна быть записана в организационном блоке, из которого могут быть вызваны также и другие блоки. Событие с более высоким приоритетом прерывает выполняемую программу в организационном блоке с более низким приоритетом. Пользователь может вызывать прерывание процесса выполнения программы событиями с высоким приоритетом, используя системные функции.

5.8.1. Типы прерываний

SIMATIC S7 предоставляет пользователю следующие типы прерываний:

- аппаратные прерывания (hardware interrupts) - прерывания, вызываемые модулем (посредством входного сигнала от процесса, или посредством сигнала, сгенерированного собственно в модуле);
- таймерные прерывания (watchdog interrupts) - прерывания, генерируемые операционной системой с заданным периодом (интервалом);
- временные (по времени суток) прерывания (time-of-day interrupts) - прерывания, генерируемые операционной системой в заданное время суток (однократные или периодические);

- прерывания с задержкой обработки (time-delay interrupts) - прерывания, генерируемые по истечении заданного периода времени (при этом в системной функции можно определять момент начала отсчета времени заданного периода);
- прерывания мультипроцессорного режима (multiprocessor interrupts) - прерывания, генерируемые другими CPU в подсети, при мультипроцессорном режиме.

Другие прерывания вызываются такими событиями, как синхронные ошибки, которые могут происходить при сканировании программы, и асинхронные ошибки, которыми могут являться, например, диагностические прерывания. Обработка этих событий обсуждается в главе "Обработка ошибок".

Событие с более высоким **приоритетом** прерывает выполняемую программу обработки некоторого события с более низким приоритетом. Основная программа (main program) имеет самый низкий приоритет (она относится к приоритетному классу 1), асинхронные ошибки имеют высокий приоритет (относятся к приоритетному классу 26). Все остальные события относятся к приоритетным классам, занимающим промежуточное положение в ряду приоритетов. В системах S7-300 приоритеты фиксированы; в системах S7-400 Вы можете самостоятельно изменять приоритеты при параметризации CPU.

Выполнение организационных блоков, запускаемых для обработки отдельных событий при сканировании программы, может быть заблокировано (disable) или, наоборот, разрешено (enable) с помощью системных функций SFC 39 (DIS_IRT) и SFC 40 (EN_IRT) соответственно, а может также быть заблокировано на время (т.е., отложено), а затем вновь разблокировано с помощью системных функций SFC 41 (DIS_AIRT) и SFC 42 (EN_AIRT) соответственно.

Нужно учитывать, что при выполнении программы обработки прерываний Вы должны иметь дело с текущими состояниями сигналов от I/O модулей (а не с состояниями сигналов входов, которые обновляются в начале выполнения основной программы), и должны записывать состояния выбранных сигналов непосредственно в I/O (не дожидаясь, пока таблица выходов образа процесса будет обновлена по окончании обработки основной программы).

В случае использования нескольких входов и выходов I/O модулей при выполнении программы, обрабатывающей прерывание, для доступа к I/O модулям достаточно использовать операции загрузки и передачи. В таком случае рекомендуется установить строгое разграничение между главной программой и программой обработки прерывания в отношении I/O сигналов.

Если необходимо обрабатывать множество входных и выходных сигналов в программе обработки прерывания, то решением для систем S7-400 может быть использование нескольких образов подпроцессов (subprocess image).

В системах S7-300 локальные стеки классов приоритетов имеют фиксированную длину - 256 байтов. В системах S7-400 Вы можете сами определять размер области этих локальных данных для каждого приоритетного класса при параметризации CPU (параметр "локальные" блока), при этом в каждом случае нельзя превышать некоторый максимальный размер, определяемый типом CPU.

Необходимо отметить, что для временных локальных данных для каждого используемого приоритетного класса должно отводиться определенное минимальное число байтов, а именно 20 байтов, для размещения стартовой информации. Для неиспользуемых приоритетных классов Вы должны определить нулевое значение.

5.8.2. Аппаратные прерывания

Аппаратные прерывания (hardware interrupts) используются для немедленного обнаружения из пользовательской программы событий, происходящих в управляемом процессе, что дает возможность выполнить программу обработки прерывания в качестве реакции на конкретное событие. STEP 7 предоставляет организационные блоки с OB 40 по OB 47 для обслуживания аппаратных прерываний. Тем не менее, то, какие из данных восьми организационных блоков доступны, зависит от типа CPU. Обработка аппаратных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). С помощью системных функций SFC 55 (WR_PARM), SFC 56 (WR_DPARM) и SFC 57 (PARM_MOD) Вы можете задавать (изменять) параметры модулей даже в рабочем режиме RUN, используя возможности аппаратных прерываний.

Аппаратные прерывания генерируются в модулях, в которых заложена такая возможность. Это может быть, например, дискретный входной модуль, который обнаруживает сигнал, поступающий от процесса, или функциональный модуль, который генерирует аппаратное прерывание, причиной которого являются процессы, происходящие в модуле. Наиболее распространены такие прерывания в аналоговых входных модулях.

В установках по умолчанию аппаратные прерывания заблокированы. Для разрешения генерации аппаратного прерывания используется параметр, являющийся статическим; Вы можете также

определить, будет ли генерироваться прерывание при приходящем событии, при уходящем событии или и при том, и при другом варианте (динамический параметр). Динамические параметры - это такие параметры, которые Вы можете изменять во время выполнения программы, используя системные функции SFC.

Аппаратное прерывание подтверждается в модуле, когда заканчивается выполнение соответствующего организационного блока с программой обслуживания прерывания.

Если во время выполнения в **S7-300** организационного блока с программой обработки аппаратного прерывания происходит событие, которое инициирует генерацию такого же аппаратного прерывания, то данное аппаратное прерывание будет потеряно, если событие, вызывающее это аппаратное прерывание, не будет больше иметь место после того, как придет подтверждение завершения обработки предыдущего прерывания. Не имеет значения, в каком модуле происходит данное событие - в модуле, генерировавшем аппаратное прерывание, которое в текущий момент было обработано, или в другом модуле.

Во время обработки аппаратного прерывания может возникнуть диагностическое прерывание. Если в том же канале в промежутке времени между моментом, когда было создано первое аппаратное прерывание, и моментом, когда оно было подтверждено (квитировано), возникает другое аппаратное прерывание, то это второе прерывание будет потеряно, что подтвердит диагностическое прерывание, которое будет сгенерировано диагностическими средствами системы.

Если во время выполнения в **S7-400** организационного блока с программой обработки аппаратного прерывания в том же канале в том же модуле происходит событие, которое инициирует генерацию такого же аппаратного прерывания, то это новое аппаратное прерывание будет потеряно. Если событие происходит в том же модуле, но в другом канале, или же в другом модуле, то операционная система перезапустит организационный блок для обработки второго прерывания, как только будет завершена обработка первого прерывания.

Начальный адрес модуля, который инициировал аппаратное прерывание, находится в байтах 6 и 7 области стартовой информации организационного блока, обрабатывающего данное прерывание. Если этот адрес является входным, то байт 5 области стартовой информации содержит B#16#54, в противоположном случае он содержит B#16#55.

Если модуль, указанный в запросе, является дискретным входным модулем, то байты с 8 по 11 содержат информацию о состоянии входов; для всех других типов модулей в этих байтах содержится информация о состоянии (статусе) прерывания в модуле.

При выполнении программы запуска (start-up program), модули не инициируют аппаратных прерываний. Обработка прерываний начинается с переходом к рабочему режиму RUN. Любые аппаратные прерывания, возникающие во время такого перехода теряются.

Если инициировано аппаратное прерывание, для которого в пользовательской программе нет организационного блока, операционная система вызывает OB 85 (организационный блок обработки ошибок, возникающих при выполнении программы). Аппаратное прерывание квитируется (подтверждается). Если OB 85 не запрограммирован, CPU переходит в режим STOP.

Обработка аппаратных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties (Правка -> Свойства объекта)* и в диалоговом окне выберите вкладку "Interrupts" ("Прерывания").

Для систем S7-300 для блока OB 40 установленный по умолчанию приоритет 16 не может быть изменен. Для систем S7-400 для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать приоритет в диапазоне от 2 до 24; при этом при установлении для блока приоритета с нулевым значением (0) блок перестанет выполняться. Никогда не назначайте один и тот же приоритет дважды, так как при этом могут быть потеряны прерывания в случае, если больше, чем 12 прерываний с одинаковым приоритетом будут инициированы одновременно.

Вы также должны разблокировать (enable) запуск аппаратных прерываний при использовании соответствующих модулей. Для этого эти модули параметризуются точно также как CPU.

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data (Системные данные)* в пользовательской папке *Blocks*, откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Данные параметризации для CPU вступают в силу немедленно после загрузки; назначения параметров для модулей вступают в силу после последующего перезапуска.

5.8.3. Таймерные прерывания

Таймерное прерывание (watchdog interrupt) - это такое прерывание, которое генерируется через периодический временной интервал и инициирует запуск организационного блока обработки данного прерывания.

Таймерные прерывания позволяют Вам организовывать периодический запуск отдельной программы, независимо от времени обработки циклической (основной) программы. В STEP 7 предусмотрены организационные блоки с OB 30 по OB 38 специально для обслуживания таймерных прерываний. Тем не менее, то, какие из данных девяти организационных блоков доступны, зависит от типа CPU.

Обработка таймерных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования) при параметризации CPU.

В S7-300 для обработки таймерного прерывания служит организационный блок OB 35, имеющий приоритет 12. При параметризации CPU Вы можете задать значение интервала времени для таймерного прерывания в диапазоне от 1 миллисекунды до 1 минуты с шагом, равным 1 миллисекунде.

При параметризации CPU **S7-400** Вы можете задать параметры таймерного прерывания. Таймерное прерывание имеет три параметра: интервал, фазовое смещение и приоритет. Вы можете определить все три параметра. Значения, которые Вы можете задать для первых двух из указанных параметров, лежат в диапазоне времени от 1 миллисекунды до 1 минуты с шагом, равным 1 миллисекунде. Значения для приоритета выбираются из диапазона значений от 2 до 24, а также возможно нулевое значение (0), в зависимости от CPU. Если приоритет приравнивается нулю, то таймерное прерывание не будет активно.

Как видно из рисунка. 5.11, для временного интервала 1 параметр "фазовое смещение" не устанавливается (или равен 0); временной интервал 2 имеет в два раза большую величину, чем временной интервал 1. Так как временной интервал 2 имеет ненулевое фазовое смещение, то организационные блоки для временного интервала 2 и организационные блоки для временного интервала 1 не могут быть вызваны одновременно. Следовательно, OB с более низким приоритетом не будет принужден ожидать, пока выполнится OB с более высоким приоритетом, и будет обрабатываться в свой срок, точно выдерживая заданный временной интервал.

Таймерные прерывания не могут быть обработаны в OB запуска (start-up). Генерация прерываний с заданными временными интервалами не начнется, пока не будет включен рабочий режим (RUN).

Если вновь генерируется то же самое таймерное прерывание, для которого в пользовательской программе уже выполняется соответствующий организационный блок, то операционная система вызывает OB 80 (организационный блок обработки временных ошибок [timing error]). Если OB 80 не запрограммирован, CPU переходит в режим STOP.

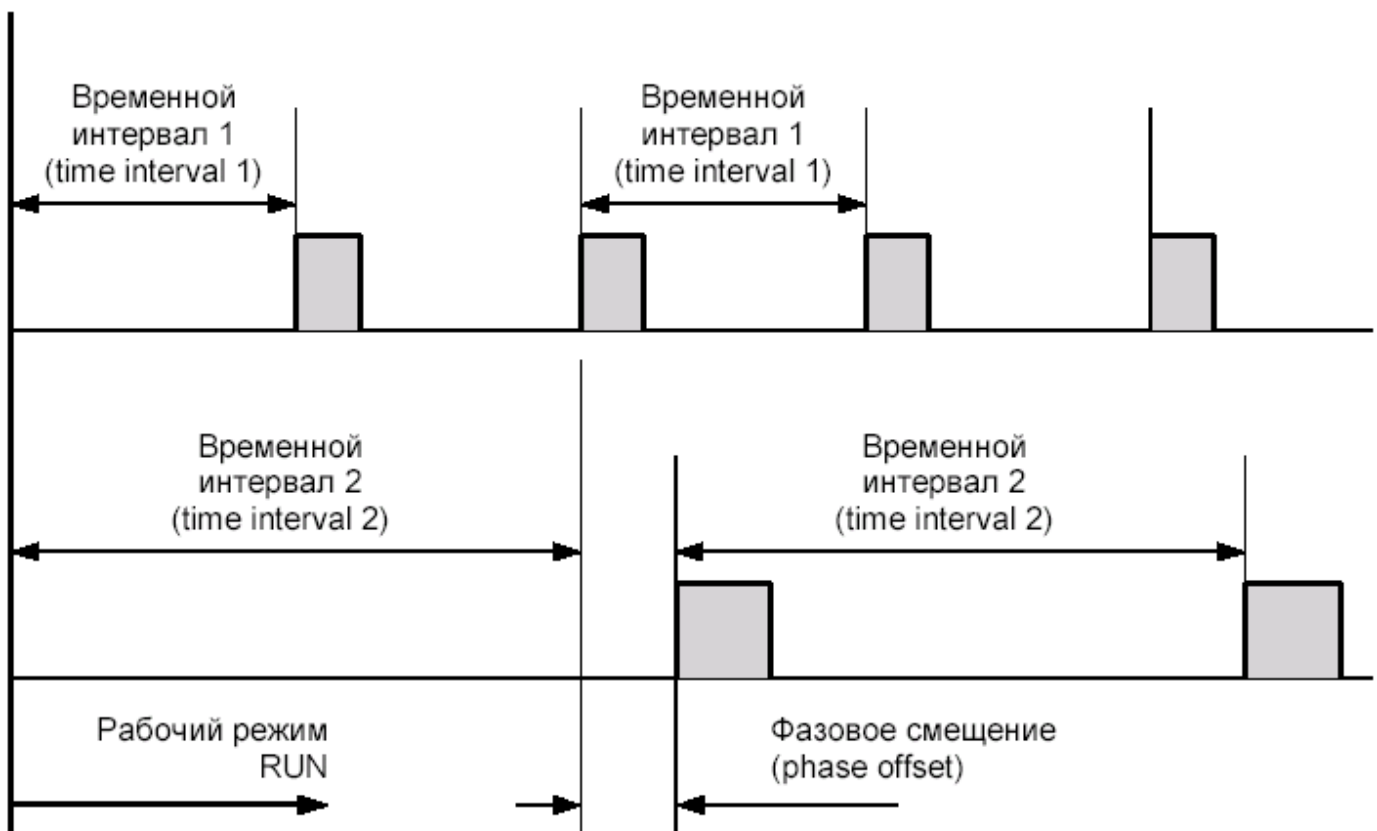


Рис. 5.11. Пример, иллюстрирующий применение параметра "фазовое смещение" (phase offset)

Операционная система сохраняет таймерное прерывание, которое не было обработано, чтобы обработать его позднее. При этом на каждый приоритетный класс может быть сохранено только одно необработанное таймерное прерывание независимо от того, как много накоплено необработанных таймерных прерываний.

Таймерное прерывание, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не может быть обработано, даже если доступен соответствующий организационный блок. В этом случае CPU переходит в режим STOP.

Обработка таймерных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties* (*Правка -> Свойства объекта*) и в диалоговом окне выберите вкладку "Cyclic Interrupts" ("Циклические прерывания").

Интервалы для каждого организационного блока выбираются в разделе "Execution" ("Выполнение"), а моменты отложенного запуска (т.е., величины временной задержки) - выбираются в разделе "Phase Offset" ("Фазовое смещение").

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data* (*Системные данные*) в пользовательской папке *Blocks*. Эти данные вступают в силу немедленно после загрузки.

5.8.4. Прерывания по времени суток

Прерывания по времени суток (time-of-day interrupts) - это такое прерывание, которое используется, когда нужно выполнять программу в заданное время суток однократно или периодически, например, ежедневно. В STEP 7 предусмотрены организационные блоки с OB 10 по OB 17 специально для обслуживания прерывания по времени суток (time-of-day interrupts). Тем не менее, то, какие из данных восьми организационных блоков доступны, зависит от типа CPU.

Обработка прерываний по времени суток может быть задана при конфигурировании оборудования (в данных конфигурирования оборудования) или же может быть выполнена посредством использования системных функций во время рабочего режима RUN. Предпосылкой правильного выполнения обработки прерываний по времени суток является корректная установка часов реального времени CPU (real-time clock).

Для запуска прерывания по времени суток Вы должны сначала установить время запуска (start time), а затем активировать прерывание. Вы можете инициировать прерывания по времени суток отдельно посредством конфигурирования оборудования или посредством использования системных функций. Необходимо отметить, что при активации посредством конфигурирования оборудования прерывание запускается автоматически после параметризации CPU. Вы можете запустить прерывание по времени суток двумя способами:

- однократный запуск, соответствующий организационный блок вызывается один раз в заданное время;
- периодический запуск в зависимости от значений параметров OB вызывается каждую минуту, ежедневно, еженедельно, ежемесячно или ежегодно.

После однократного запуска OB прерывания по времени суток данное прерывание отменяется. Вы также можете отменить прерывание по времени суток, используя системную функцию SFC 29 (CAN_TINT).

Если Вы вновь хотите возобновить отмененное прерывание по времени суток, то Вы должны будете вновь установить параметр "время запуска" (start time), а затем вновь активировать прерывание. Вы также можете запросить о состоянии прерывания по времени суток, используя системную функцию SFC 31 (QRY_TINT).

Во время "холодного" (cold) перезапуска или полного (complete) перезапуска операционная система стирает все установки, сделанные посредством SFC. Установки, выполненные посредством конфигурирования оборудования сохраняются. При "теплом" (warm) перезапуске CPU продолжает обработку прерываний по времени суток в первом завершенном (complete) цикле сканирования основной программы.

Вы можете запросить о состоянии прерываний по времени суток, вызывая системную функцию SFC 31 в OB запуска, и последовательно отменяя или вновь инициализируя и вновь активируя прерывания. Прерывания по времени суток обрабатываются только в рабочем режиме RUN.

Если вызывается OB обработки прерывания по времени суток, но данный OB не запрограммирован, то операционная система вызывает OB 85 (организационный блок обработки ошибок, возникающих при выполнении программы). Если OB 85 не запрограммирован, CPU переходит в режим STOP.

Прерывание по времени суток, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не может быть обработано, даже если доступен соответствующий организационный блок. В этом случае CPU переходит в режим STOP.

Если Вы активировали прерывание по времени суток в режиме однократного вызова, и если при этом заданное стартовое время (start time) уже прошло на текущий момент (по часам реального времени системы), то операционная система вызовет OB 80 (организационный блок обработки временных ошибок. Если OB 80 недоступен, то CPU перейдет в режим STOP.

Если Вы активировали прерывание по времени суток в режиме периодического вызова, и если при этом заданное стартовое время (start time) уже прошло на текущий момент (по часам реального времени системы), то операционная система сгенерирует прерывание по времени суток в следующем периоде в заданный момент времени (согласно заданным временным параметрам).

Если Вы перевели часы реального времени вперед с целью коррекции их показаний или для синхронизации так, что при этом проскочили заданное стартовое время прерывания по времени суток, то операционная система вызовет OB 80 (организационный блок обработки временных ошибок). После этого OB обработки прерывания по времени суток будет вызван только один раз.

Если Вы перевели часы реального времени назад с целью коррекции их показаний или для синхронизации, то активированный OB обработки прерывания по времени суток не будет в дальнейшем повторно выполняться в моменты времени, которые ранее уже были пройдены до перевода часов назад.

Если вызывается OB обработки прерывания по времени суток, для которого в пользовательской программе уже выполняется соответствующий организационный блок (режим периодического запуска), то операционная система вызывает OB 80. Когда OB 80 и OB обработки прерывания по времени суток будут выполнены, OB обработки прерывания по времени суток будет перезапущен.

Обработка прерываний по времени суток может быть задана при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties* (*Правка -> Свойства объекта*) и в диалоговом окне выберите вкладку "Time-of-Day" ("Время суток").

Для контроллеров S7-300 для блока обработки установлен по умолчанию фиксированный приоритет 2. Для систем S7-400 для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать значение приоритета из диапазона от 2 до 24 или равное 0; при этом при установлении для блока приоритета со значением 0 блок перестанет выполняться.

Опция "Active" ("Активировано") автоматически включает запуск прерывания по времени суток. Опция "Execution" ("Выполнение") включает отображение списка, который позволит Вам выбрать вариант выполнения прерывания по времени суток - в режиме однократного вызова или в режиме периодического вызова. Последний параметр определяет время выполнения прерывания по времени суток - в нем задается "стартовое время" или время запуска прерывания (дата и время).

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data* в автономной пользовательской программе *Blocks*. Эти данные вступают в силу немедленно после загрузки.

Для управления прерываниями по времени суток предусмотрены следующие системные функции:

- SFC 28 (SET_TINT) - функция для установки прерывания по времени суток;
- SFC 29 (CAN_TINT) - функция для отмены прерывания по времени суток;
- SFC 30 (ACT_TINT) - функция для активации прерывания по времени суток;
- SFC 31 (QRY_TINT) - функция для запроса о состоянии прерывания по времени суток.

5.8.5. Прерывания с задержкой обработки

Прерывания с задержкой обработки (time-delay interrupts) - это такое прерывание, которое позволяет обеспечить выполнение функции таймера задержки без использования стандартных функций таймера. В STEP 7 предусмотрены организационные блоки с OB 20 по OB 23 специально для обслуживания прерывания с задержкой обработки. Тем не менее, то, какие из данных четырех организационных блоков доступны, зависит от типа CPU.

Приоритеты OB обработки прерываний с задержкой обработки задаются при конфигурировании оборудования (в данных конфигурирования оборудования).

Прерывание с задержкой обработки запускается при вызове системной функции SFC 32 (SRT_DINT) в момент программного события; данная системная функция передает операционной системе значение временного интервала и номер выбранного организационного блока. После того, как истекает заданный временной интервал, происходит вызов определенного организационного блока.

Вы можете отменить обслуживание прерывания с задержкой обработки посредством системной функции SFC 33 (CAN_DINT), в результате чего связанный с этим прерыванием OB не будет более вызываться.

Вы можете также запросить информацию о состоянии прерывания с задержкой обработки, используя системную функцию SFC 34 (QRY_DINT).

Во время "холодного" или полного перезапуска операционная система стирает все установки, сделанные для прерывания с задержкой обработки. При "теплом" перезапуске установки охраняются, пока идет обработка в рабочем режиме RUN, в то время как "оставшаяся часть" цикла считается частью подпрограммы запуска (start-up).

Вы можете запускать прерывание с задержкой обработки в подпрограмме запуска, вызывая системную функцию SFC 32. В момент, когда будет закончен временной интервал прерывания (интервал отсчета задержки) CPU должен находиться в состоянии рабочего режима RUN, для того чтобы обеспечить выполнение связанного с прерыванием организационного блока. Если это условие не выполняется, CPU будет ждать окончания выполнения подпрограммы запуска (start-up), и лишь затем вызовет для выполнения связанный с прерыванием организационный блок; все это происходит до обработки первого сегмента основной программы.

Если вызывается ОВ обработки прерывания с задержкой обработки, но данный ОВ не запрограммирован, то операционная система вызывает ОВ 85. Если ОВ 85 не запрограммирован, CPU переходит в режим STOP. Если истек временной интервал (интервал отсчета задержки) и вызывается ОВ обработки данного прерывания, для которого в пользовательской программе уже выполняется соответствующий организационный блок, то операционная система вызывает ОВ 80 или переходит в режим STOP, если ОВ 80 недоступен в пользовательской программе.

Прерывание с задержкой обработки, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не сможет быть обработано, даже если соответствующий организационный блок был запрограммирован. В этом случае CPU переходит в режим STOP.

Вызов организационных блоков обработки прерываний с задержкой обработки может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 и SFC 40 соответственно, а также задержан на время и разблокирован с помощью системных функций SFC 41 и SFC 42 соответственно.

Обработка прерываний с задержкой обработки может быть задана при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: Edit -> Object Properties (Правка -> Свойства объекта) и в диалоговом окне выберите вкладку "Interrupts" ("Прерывания").

Для контроллеров S7-300 (за исключением CPU 318) блок обработки прерывания с задержкой имеет по умолчанию фиксированный приоритет 3. Для систем S7-400 (а также для CPU 318) для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать значение приоритета из диапазона от 2 до 24 или равное 0; при этом при установлении для блока приоритета со значением 0 блок перестанет выполняться.

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в System data (Системные данные) в автономной пользовательской программе Blocks (Блоки), откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Эти данные вступают в силу немедленно после загрузки.

Для управления прерываниями с задержкой обработки предусмотрены следующие системные функции:

- SFC 32 (SRT_DINT) - функция для установки задержки запуска ОВ прерывания;
- SFC 33 (CAN_DINT) - функция для отмены прерывания с задержкой обработки;
- SFC 34 (QRY_DINT) - функция для запроса о состоянии прерывания с задержкой обработки.

5.8.6. Прерывание мультипроцессорного режима

Прерывание мультипроцессорного режима позволяет обеспечить синхронную реакцию на событие во всех CPU, работающих в мультипроцессорном режиме. Прерывание мультипроцессорного режима запускается с помощью SFC 35 (MP_ALM) Для обслуживания прерывания мультипроцессорного режима предусмотрен организационный блок ОВ 60, для которого установлен фиксированный приоритет, равный 25.

Вызов системной функции SFC 35 вызывает выполнение организационного блока ОВ 60. Если CPU работает в однопроцессорном режиме, то ОВ 60 запускается немедленно. Если режим работы мультипроцессорный, то ОВ 60 запускается одновременно на всех CPU мультипроцессорной системы, что означает, что CPU, в котором вызвана функция SFC 35 отложит запуск блока ОВ 60 до момента, когда все другие CPU сообщат о своей готовности.

Обработка прерываний мультипроцессорного режима не задаётся при конфигурировании оборудования (в данных конфигурирования оборудования) - эта функция заложена в каждом CPU, предназначенном для работы в мультипроцессорном режиме. Тем не менее, не смотря на это, минимально

необходимое количество байтов в области локальных данных (по крайней мере, 20 байтов) должно быть все же зарезервировано на вкладке "Local Data" CPU для приоритетного класса 25.

Прерывание мультипроцессорного режима запускается только в рабочем режиме RUN. Вызов системной функции SFC 35 при выполнении подпрограммы запуска заканчивается с возвращением кода ошибки 32929 (W#16#80A1) в качестве её значения.

В случае, если вызывается функция SFC 35 в то время, когда OB 60 все еще обрабатывается, системная функция возвращает код ошибки 32928 (W#16#80A0) в качестве своего значения. И при этом организационный блок OB 60 не запускается ни на одном CPU.

Если блок OB 60 недоступен в одном из CPU во время вызова этого организационного блока или блокирован, или задержано его выполнение посредством системных функций, то функция SFC 35 не вызывает никаких действий в данном CPU и не сообщает при этом об ошибке.

Вызов организационного блока обработки прерывания с задержкой обработки может быть заблокирован и разблокирован с помощью системных функций SFC 39 и SFC 40 соответственно, а также задержан на время и разблокирован с помощью системных функций SFC 41 и SFC 42 соответственно.

5.9. Параметры перезапуска

5.9.1. Режимы работы

После включения электропитания перед началом выполнения основной программы (main program) CPU выполняет подпрограмму перезапуска (restart routine). Режим запуска START-UP (ЗАПУСК) является одним из режимов работы CPU, наряду с режимами STOP (СТОП) и RUN (РАБОТА). Эта глава описывает поведение CPU при переходе от режима START-UP и, наоборот, к режиму START-UP, а также при выполнении собственно подпрограммы перезапуска (restart routine).

После включения электропитания CPU находится в режиме STOP. Если на передней панели CPU переключатель стоит в положении RUN или в положении RUN-P, то CPU переключается в режим START-UP, а после этого - в режим RUN. Если происходит "неисправимая" ошибка, пока CPU находится в режиме START-UP или RUN, или если Вы переключите переключатель на передней панели CPU в положение STOP, то CPU переключается в режим STOP.

Пользовательская программа может быть протестирована в пошаговом режиме с использованием "точек прерывания" в рабочем режиме HOLD (ПАУЗА). Вы можете переключаться в этот режим и из режима RUN, и из режима START-UP, а затем, после окончания тестирования программы, Вы можете переключиться опять в исходные режимы. Вы также можете переключить CPU в режим STOP из режима HOLD.

При параметризации CPU на вкладке "Restart" ("Перезапуск") Вы можете определять параметры перезапуска, такие как максимально разрешенное количество времени для сигналов "Ready" ("Готов") от модулей после включения питания, а также указывать, должен ли CPU запускаться, если данные конфигурации не согласуются с "фактической конфигурацией", или определять режим перезапуска CPU. SIMATIC S7 предоставляет пользователю следующие три режима перезапуска, а именно: "холодный" перезапуск (*cold restart*), "полный" перезапуск (*complete restart*) и "теплый" перезапуск (*warm restart*). При "холодном" и "полном" перезапусках основная программа всегда выполняется с начала. При "теплом" перезапуске основная программа продолжает выполняться с той точки, в которой ее выполнение было прервано, и она "заканчивает" прерванный цикл выполнения. Другие прерывания вызываются такими событиями, как синхронные ошибки, которые могут происходить при сканировании программы, и асинхронные ошибки, которыми могут являться, например, диагностические прерывания. Обработка этих событий обсуждается в разделе "Обработка ошибок".

В S7 CPU выпуска до октября 1998 г. предусмотрены "полный" перезапуск (*complete restart*) и "теплый" перезапуск (*warm restart*). При этом функционально "полный" перезапуск соответствует "теплому" перезапуску.

Вы можете однократно просканировать подпрограмму запуска START-UP. Специально для этих целей в STEP 7 имеются организационные блоки - OB 102 (для "холодного" перезапуска), OB 100 (для "полного" перезапуска) и OB 101 (для "теплого" перезапуска). В блоках уже проведена параметризация модулей кроме CPU и запрограммированы установки "по умолчанию" для Вашей основной программы.

CPU переключается в режим HOLD (ПАУЗА) в случае, когда Вы тестируете пользовательскую программу с использованием "точек прерывания" (в пошаговом режиме). При этом должен светиться светодиодный индикатор "STOP", а индикатор "RUN" должен мигать. В режиме HOLD выходные модули заблокированы. Запись в эти модули приводит к изменению состояний сигналов в памяти, но не переключает состояний сигналов на выходах модулей. При этом модули не могут быть вновь разблокированы, пока не будет выключен режим HOLD.

В режиме HOLD все процессы, в которых присутствует функция измерения времени, не функционируют (кроме функции часов реального времени). К этим функциям относятся таймеры, тактовые меркеры, счетчики отработанного времени, функции отслеживания времени цикла и минимального времени цикла сканирования, а также обслуживание прерывания по времени суток и прерывания с задержкой обработки.

Исключение: функция часов реального времени в этом режиме продолжает нормально работать. При каждом последующем переходе к следующему оператору в программе, обрабатываемой в режиме "пошагового" ("single step") тестирования, значения таймеров инкрементируются в соответствии с величиной шага - таким образом имитируется динамическое поведение программы при "обычном" ее сканировании.

В режиме HOLD CPU способен быть "пассивным" коммуникационным партнером, что означает, что он может принимать глобальные данные или принимать участие в одностороннем обмене данными.

При сбое электропитания в то время, когда CPU находится в режиме HOLD, CPU с резервной батареей питания при восстановлении нормального электропитания переходят в режим STOP, тогда как CPU без резервной батареи автоматически выполняют полный перезапуск.

В режимах STOP и HOLD модули заблокированы (заблокированы выходы, OD-сигнал).

Заблокированные модули имеют на своих выходах "нулевой" сигнал (сигнал нулевой величины) или, если они оснащены такой функцией, то имеют на своих выходах так называемый "сигнал замещения" (или значение замещения). С помощью таблицы переменных Вы можете управлять выходами модулей, используя функцию "Enable Peripheral Outputs" ("Разблокировать периферийные выходы") даже в режиме STOP.

Во время перезапуска выходные модули остаются в заблокированном состоянии. Только после того, как начинается цикл сканирования основной программы, выходные модули разблокируются. Состояния сигналов из памяти модулей (но не в области образа процесса!) поступают на выходы.

В то время, пока выходные модули остаются в заблокированном состоянии, отдельные биты памяти модуля могут быть установлены или посредством прямого доступа (с помощью операторов передачи [transfer] с адресацией области PQ), или посредством передачи таблицы выходов образа процесса. Если CPU отменяет сигнал блокировки (Disable), то состояния сигналов из памяти модуля поступают на внешние выходы модуля.

При "холодном" перезапуске (OB 102) и "полном" перезапуске (OB 100) данные образа процесса и данные в памяти модуля стираются. Если необходимо сканировать входы в OB 102 или в OB 100, Вы сначала должны загрузить состояния сигналов из модуля, используя прямой доступ. После этого Вы можете устанавливать входы (переносить их, например, с помощью операторов загрузки из адресной области PI в адресную область I), а затем работать со входами. Если необходимо установить определенные выходы при переходе от полного перезапуска к циклическому сканированию программы (перед вызовом OB 1), то Вы должны использовать прямой доступ при адресации выходных модулей. В данном случае не достаточно только лишь установить выходы (в области образа процесса), так как таблица выходов образа процесса не пересылается в модули в конце подпрограммы полного перезапуска.

При "теплом" перезапуске "старые" таблицы выходов и входов образа процесса, которые были действительными на момент выключения электропитания установки или при переводе ее в режим STOP, используются в OB 101 и с момента возобновления до окончания цикла сканирования программы. В конце этого цикла таблица выходов образа процесса пересылается в память модуля (но при этом еще не вызывает изменение выходных сигналов модуля на его внешних выходах, так как в это время выходные модули пока еще находятся в заблокированном состоянии).

Вы можете провести параметризацию CPU, чтобы очистить таблицу выходов образа процесса и память модуля в конце процедуры "теплого" перезапуска. Перед переходом к обработке OB 1 CPU отменяет сигнал блокировки, после этого состояния сигналов из памяти модуля передаются на его внешние выходы.

При "холодном" перезапуске CPU вызывает организационный блок OB 102; при "полном" перезапуске CPU вызывает организационный блок OB 100. При отсутствии блока OB 100 или блока OB 102 CPU немедленно начинает выполнять цикл сканирования программы.

При "теплом" перезапуске CPU однократно вызывает организационный блок OB 101 перед началом обработки основной программы. При отсутствии организационного блока OB 101 CPU продолжает выполнять сканирование программы с точки прерывания ее выполнения в последний раз. Стартовая информация во временных локальных данных имеет одинаковый формат для всех организационных блоков.

Причина перезапуска указывается в запросе на перезапуск (байт 1):

- V#16#81 - ручной режим "полного" перезапуска (OB 100),
- V#16#82 - автоматический режим "полного" перезапуска,
- V#16#83 - ручной режим "теплого" перезапуска (OB 101),
- V#16#84 - автоматический режим "теплого" перезапуска,
- V#16#85 - ручной режим "холодного" перезапуска (OB 102),

- В#16#86 - автоматический режим "холодного" перезапуска.

Номер события остановки и дополнительная информация более точно определяют событие перезапуска (они сообщают, например, о том, был ли ручной "холодный" перезапуск инициирован с помощью переключателя режимов). С помощью этой информации Вы можете соответствующим образом настроить обусловленную событиями подпрограмму перезапуска.

CPU переходит в режим STOP в следующих случаях:

- когда CPU выключается;
- когда переключатель режимов переключается с режима RUN (РАБОТА) в режим STOP (СТОП);
- когда во время сканирования программы происходит "неустраняемая" ошибка;
- когда выполняется системная функция SFC 46 STP;
- когда приходит запрос на остановку CPU от коммуникационной функции (запрос на переход в режим STOP от программатора или коммуникационного функционального блока от другого CPU).

CPU записывает причину перехода в режим STOP в диагностический буфер. Находясь в данном режиме, Вы можете также прочитать информацию CPU с помощью программатора, чтобы локализовать источник возможной проблемы.

В режиме STOP пользовательская программа не сканируется. CPU восстанавливает установки: или значения, которые Вы задавали при конфигурировании оборудования при параметризации CPU, или установки, принимаемые по умолчанию, и устанавливает модули в определенное исходное состояние.

В режиме STOP CPU может принимать глобальные данные посредством GD-коммуникаций и выполнять односторонний "пассивный" обмен данными. В режиме STOP часы реального времени продолжают функционировать.

Вы можете параметризовать CPU в режиме STOP, например, Вы можете установить MPI-адрес, переслать или модифицировать программу пользователя или выполнить сброс памяти CPU.

Сброс памяти (Memory Reset) приводит CPU к исходному состоянию. Вы можете инициировать сброс памяти, используя программатор только в режиме STOP или при помощи переключателя режимов работы. Для этого необходимо удерживать в положении MRES по крайней мере 3 секунды, затем - отпустить максимум на 3 секунды и вновь удерживать в положении MRES по крайней мере 3 секунды.

CPU при сбросе памяти стирает пользовательскую программу целиком - и в рабочей и в загрузочной RAM-памяти. Информация в системной памяти (меркеры, таймеры и счетчики) также стирается независимо от установок ретентивности.

CPU при сбросе памяти устанавливает параметры всех модулей (включая и свои собственные) в состояния, принятые для них как "значения по умолчанию". Как исключение, только параметры для MPI-подсети не изменяются, поэтому к CPU, память которого была "сброшена", все еще остается возможным доступ посредством MPI-шины. Также сброс памяти не влияет на диагностический буфер, на часы реального времени и измерители времени наработки (часы учета рабочего времени).

Если используется модуль памяти Flash EPROM, то CPU копирует пользовательскую программу из модуля памяти в рабочую память. CPU также копирует все данные конфигурации, которые он сможет найти в модуле памяти.

Область памяти является ретентивной, если ее содержимое сохраняется, при выключении источника питания, также как и при переходе от режима STOP (СТОП) к режиму RUN (РАБОТА) после включения электропитания (при "холодном" перезапуске и при "теплом" перезапуске).

Ретентивной область памяти может быть для меркеров, таймеров, счетчиков, для S7-300 и для блоков данных. Размер области данных, которая может быть объявлена ретентивной, зависит от типа CPU. Вы можете определить области данных как ретентивные на вкладке "Retentivity" ("Ретентивность") при параметризации CPU.

Установки ретентивности сохраняются в блоках системных данных (SDB) в загрузочной памяти, то есть в модуле памяти. Если модуль памяти имеет тип RAM, то для постоянного сохранения там данных Вы должны использовать резервную батарею.

При работе с резервной батареей состояния всех сигналов меркеров, таймеров, счетчиков, сохраняются. Пользовательская программа и данные пользователя также остаются сохраненными. При наличии резервной батареи питания не играет роли тип используемого модуля памяти - RAM или флэш-EPROM.

Если используется модуль памяти Flash EPROM и отсутствует резервная батарея питания, то S7-300 и S7-400 ведут себя по-разному. В контроллерах S7-300 состояния сигналов меркеров, таймеров, счетчиков, объявленных ретентивными, сохраняются, тогда как в S7-400 - не сохраняются.

Содержание ретентивных блоков данных также остается неизменным в S7-300. Необходимо отметить, что в S7-300 содержание ретентивных областей данных сохраняется в CPU, а не в модуле памяти.

Остальные блоки данных в контроллерах S7-300 и все блоки данных в контроллерах S7-400 копируются из модуля памяти в рабочую память как кодовые блоки. Единственные блоки, содержание которых сохраняется, это блоки, находящиеся в модуле памяти. Блоки данных, которые создаются с

помощью системной функции SFC 22 (CREAT_DB), не являются реманентными. После перезапуска блоки данных имеют содержание, которое хранится в модуле памяти, т.е. то самое содержание, с которым они были запрограммированы.

5.9.4 Определение параметров для перезапуска

При параметризации CPU на вкладке "Startup" ("Запуск") Вы можете определить режим перезапуска, сделав следующие установки.

- "Restart when the set configuration is not the same as the actual configuration" (перезапуск когда набор параметров конфигурации не совпадает с фактической конфигурацией). Перезапуск выполняется, если конфигурация параметров оборудования не согласуется с фактической конфигурацией.
- "Hardware test at complete restart (warm restart)" (тестирование оборудования при "полном" перезапуске (при "теплом" перезапуске)). S7-300 CPU выполняют тестирование оборудования при включении электропитания.
- "Delete PIQ at warm restart" (очищать область таблицы выходов образа процесса при "теплом" перезапуске). S7-400 CPU очищают области всех таблиц выходов образа процесса при "теплом" перезапуске.
- "Disable warm restart at manual restart" (запретить "теплый" перезапуск при ручном режиме перезапуска). "Теплый" перезапуск в ручном режиме запрещен.
- "Restart following POWER UP" (перезапуск после включения питания). Определение типа перезапуска после включения питания.
- "Monitoring time for ready signal of the modules" (значение времени мониторинга, отведенного модулям на посылку сигнала готовности). Если истекло время, отведенное модулям на посылку сигнала готовности, CPU возвращается к режиму STOP; событие записывается в диагностический буфер (особенно важно при включении питания в стойках расширения).
- "Monitoring time for transferring the parameters to the modules" (значение времени мониторинга, отведенного на пересылку параметров модулям). Если истекает время, отведенное на пересылку параметров модулям, CPU возвращается к режиму STOP; событие записывается в диагностический буфер. (В случае возникновения такой ошибки Вы можете при параметризации CPU задать самое большое значение времени мониторинга для пересылки параметров модулям - без сброса памяти. Если Вы пересылаете системные данные "пустого" проекта, в котором задано новое значение времени для пересылки параметров модулям, то параметризация модулей заканчивается внутри отрезка времени, равного "старому" значению времени мониторинга).
- "Monitoring time for warm restart" (значение времени мониторинга, в течение которого сохраняется разрешение на "теплый" перезапуск). Если время между моментом выключения и моментом включения питания или между моментом перехода к режиму STOP и моментом перехода к режиму RUN больше, чем значение времени мониторинга, CPU возвращается к режиму STOP. Значение времени мониторинга, равное 0, выключает данный режим контроля времени.

5.10. Обработка ошибок

Об ошибках или сбоях, обнаруженных в модулях или собственно в CPU, CPU сообщает различными способами:

- при ошибках, возникших при выполнении арифметических операций (переполнение, некорректное число формата REAL), - посредством установки битов состояния (бит состояния OV, например, сообщает о переполнении числа);
- при ошибках, обнаруженных при выполнении пользовательской программы ("synchronous errors" - "синхронные ошибки"), - посредством вызова организационных блоков OB 121 и OB 122;
- при обнаруженных в программируемом контроллере ошибках, которые не связаны с выполнением пользовательской программы ("asynchronous errors" - "асинхронные ошибки"), - посредством вызова организационных блоков из ряда OB 80 - OB 87.

CPU сигнализирует о том, что произошла ошибка или отказ и, в некоторых случаях, указывает на причину ошибки (сбоя) посредством светоиндикаторов - светодиодов индикации ошибок, расположенных на передней панели. В случае, если произошла неисправимая ошибка (например, неверный код оператора), CPU сразу переходит в режим STOP.

Если CPU находится в режиме STOP, Вы можете использовать программатор и функции для работы с информацией CPU для считывания содержимого стека блоков (B stack), стека прерываний (I stack) и стека локальных данных (L stack), что позволит Вам сделать заключение о причинах возникновения ошибок.

Системные средства диагностики могут обнаруживать ошибки/сбои в модулях и помещать информацию о них в диагностический буфер. Информация о переходах CPU из режима в режим (также как и информация о причинах перехода CPU в режим STOP) помещается в диагностический буфер.

Содержимое диагностического буфера сохраняется при переходе системы в режим STOP, при сбросе памяти, а также при сбое в системе электропитания. Поэтому содержимое диагностического буфера может быть считано после последующего восстановления электропитания и выполнения подпрограммы перезапуска с помощью программатора. При работе с CPU новых выпусков Вы можете при параметризации CPU задавать число вводимых записей в диагностический буфер, которые должны там сохраняться.

5.10.1. Синхронные ошибки

Операционная система CPU генерирует сигнал о том, что обнаружена синхронная ошибка, если произошла ошибка, имеющая прямое отношение к сканируемой программе. В случае, если организационный блок OB обработки синхронных ошибок не запрограммирован, то CPU сразу переходит в режим STOP. Среди синхронных ошибок различаются:

- **ошибки программирования**, для обработки которых вызывается блок OB 121, и
- **ошибки программирования**, для обработки которых вызывается блок OB 122.

Блок OB для обработки синхронных ошибок имеет такой же приоритет как и блок, в котором произошла ошибка. Поэтому из OB для обработки синхронных ошибок возможен доступ к регистрам блока, в котором произошло прерывание, и также поэтому программа в этом OB для обработки синхронных ошибок (при определенных условиях с измененным содержанием) может возвращать значения регистров в блок, в котором произошло прерывание.

Необходимо отметить, что при вызове OB для обработки синхронных ошибок, 20 байтов его стартовой информации также передаются в L-стек приоритетного класса, в котором произошла ошибка, как и другие временные локальные данные OB-блока для обработки синхронных ошибок.

В случае S7-400 в OB для обработки синхронных ошибок может быть вызван по прерыванию другой OB для обработки синхронных ошибок. Глубина вложения вызовов блоков для OB для обработки синхронных ошибок составляет 3 для S7-400 CPU и 4 для S7-300 CPU.

Пользователь может запрещать и разрешать вызовы OB для обработки синхронных ошибок с помощью системных функций SFC 36 (MSK_FLT), SFC 37 (DMSK_FLT), SFC 38 (READ_ERR).

Следующие системные функции обеспечивают выполнение обработки синхронных ошибок:

- SFC 36 (MSK_FLT) служит для маскирования синхронных ошибок (обеспечивает блокирование вызовов блоков OB);
- SFC 37 (DMSK_FLT) служит для демаскирования синхронных ошибок (обеспечивает разрешение вызовов блоков OB);
- SFC 38 (READ_ERR) служит для считывания регистра ошибок.

Операционная система вносит информацию о синхронной ошибке в диагностический буфер независимо от использования системных функций SFC 36, SFC 37 и SFC 38.

Для того, чтобы управлять системными функциями, обеспечивающими обработку синхронных ошибок, используются "фильтры для ошибок" ("error filters"). В фильтре для ошибок программирования для каждой ошибки программирования предназначается один бит; в фильтре ошибок доступа также для каждой ошибки доступа предназначается один бит.

При определении фильтра ошибок Вы устанавливаете соответствующий бит, который соответствует синхронной ошибке, для которой Вы должны выполнить маскирование, демаскирование или запрос о состоянии регистра ошибок. После запроса системная функция SFC 38 возвращает "1" в битах, соответствующих синхронным ошибкам, которые все еще маскированы, или имели место.

Фильтр ошибок доступа размещен в переменной OB122_SW_FLT стартовой информации для OB 122. Фильтр ошибок программирования - в переменной OB121_SW_FLT. S7-400 CPU различают два типа ошибок доступа: обращение к несуществующему модулю и некорректное обращение к существующему модулю. Если в модуле произошел сбой во время работы, то этот модуль помечается как "несуществующий" приблизительно через 150 мс после попытки доступа к нему, и фиксируется ошибка доступа к I/O при каждой последующей попытке обращения к данному модулю.

CPU также выдает сообщение об ошибке доступа к I/O, когда совершается попытка обращения к несуществующему модулю независимо от того, выполняется попытка прямого доступа к модулю (через область I/O) или попытка косвенного доступа к модулю (через область образа процесса).

Системная функция SFC 36 позволяет блокировать вызовы OB обработки синхронных ошибок с помощью фильтров ошибок. "1" в фильтре ошибок индицирует те синхронные ошибки, при появлении которых не будут вызываться организационные блоки для их обработки (в таких случаях говорят, что синхронные ошибки "маскированы" - "masked"). При маскировании синхронных ошибок с помощью

фильтров ошибок данные маскирования добавляются к данным маскирования, которые хранятся в памяти операционной системы. Системная функция SFC 36 возвращает значение функции, равное W#16#0001, показывающее, что в сохраненных данных маскирования уже установлен хотя бы один бит из вновь устанавливаемых согласно определению входных параметров функции.

Функция SFC 36 возвращает "1" в выходных параметрах для всех маскированных ошибок на текущий момент.

Если происходит синхронная ошибка, которая ранее была маскирована, то соответствующий ей блок ОВ не вызывается, а информация об ошибке поступает в диагностический буфер. Блокировка вызова ОВ касается текущего приоритетного класса ("priority level"). Например, если Вы заблокировали вызов ОВ обработки синхронной ошибки в основной программе, данный блок ОВ все же будет вызываться в случае возникновения этой ошибки в подпрограмме другого ОВ, обслуживающего прерывание.

Системная функция SFC 37 DMSK_FLT позволяет разблокировать (разрешить) вызовы ОВ обработки синхронных ошибок с помощью фильтров ошибок. Вы должны ввести "1" в фильтр ошибок для обозначения тех синхронных ошибок, для которых вновь должны вызываться организационные блоки их обработки (в таких случаях говорят, что синхронные ошибки "демаскированы" - "unmasked").

Установленные пользователем биты соответствуют определенным битам в регистре ошибок, которые теперь будут сброшены. Системная функция SFC 37 возвращает значение функции W#16#0001, показывающее, что в уже сохраненных данных маскирования нет ни одного установленного (маскирующего соответствующую ошибку) бита, который должен быть сброшен в соответствии с вновь определенными входными параметрами для системной функции (для демаскирования соответствующей ошибки). Функция SFC 37 возвращает "1" в выходных параметрах для всех маскированных ошибок на текущий момент.

Если происходит синхронная ошибка, которая была демаскирована, то вызывается соответствующий ей блок ОВ, а информация об ошибке поступает в диагностический буфер. Блокировка (Disable) вызова ОВ касается текущего приоритетного класса.

Системная функция SFC 38 служит для считывания регистра ошибок с помощью фильтров ошибок. Вы должны ввести "1" в фильтр ошибок для обозначения тех синхронных ошибок, состояние битов которых должно быть считано. Системная функция SFC 38 возвращает значение функции W#16#0001, если в сохраненных данных маскирования не установлен (для маскирования) хотя бы один бит из тех битов, для которых производится запрос на считывание при определении входных параметров функции.

Функция SFC 36 возвращает "1" в выходных параметрах для выбранных ошибок, если эти ошибки произошли, и сбрасывает в регистре ошибок биты тех ошибок, для которых производится запрос. При этом с помощью данной функции считывается информация о синхронных ошибках, которые относятся к текущему приоритетному классу (приоритетному уровню - "priority level").

Системная функция SFC 44 (REPL_VAL) позволяет пользователю ввести "заменяющее" значение или значение замены в аккумулятор 1 из организационного блока ОВ обработки синхронной ошибки. Вы можете использовать системную функцию SFC 44, когда нет больше возможности считывать никакие значения из модуля (например, в случае, когда модуль неисправен). Если Вы запрограммировали системную функцию SFC 44, всякий раз при попытке доступа к модулю будет вызываться организационный блок ОВ 122 (блок обработки ошибки доступа). Когда Вы вызываете в ОВ122 функцию SFC 44, Вы можете загрузить значение замены в аккумулятор. При этом сканирование программы продолжится далее с данным значением замены.

5.10.2. Асинхронные ошибки

Асинхронные ошибки - это такие ошибки, которые могут произойти независимо от выполнения (сканирования) программы. Если возникает какая-либо асинхронная ошибка, операционная система вызывает один из нижеуказанных организационных блоков:

ОВ 80 - блок обработки ошибки времени (timing error);

ОВ 81 - блок обработки ошибки в блоке питания (power supply error);

ОВ 82 - блок обработки диагностического прерывания (diagnostic interrupt);

ОВ 83 - блок обработки прерывания установки/удаления модуля (insert/remove module interrupt);

ОВ 84 - блок обработки отказа аппаратной части CPU (CPU hardware fault);

ОВ 85 - блок обработки ошибки выполнения программы (program execution error);

ОВ 86 - блок обработки отказа стойки (rack failure);

ОВ 87 - блок обработки коммуникационной ошибки (communication error).

Вызов блока ОВ 82 (блок обработки диагностического прерывания) рассматривается в разделе "Системная диагностика".

Для S7-400H существуют еще три дополнительных организационных блока для обработки асинхронных ошибок:

ОВ 70 - блок обработки ошибки резервирования периферии (I/O redundancy errors);

ОВ 72 - блок обработки ошибки резервирования CPU (CPU redundancy errors);

ОВ 73 - блок обработки ошибки резервирования коммуникаций (communication redundancy errors).

Выполнение данных организационных блоков, запускаемых для обработки асинхронных ошибок, может быть заблокировано или, наоборот, разрешено с помощью системных функций SFC 39 (DIS_IRT) и SFC 40 (EN_IRT) соответственно, а может также быть заблокировано на время (отложено), а затем вновь разблокировано с помощью системных функций SFC 41 (DIS_AIRT) и SFC 42 (EN_AIRT) соответственно.

Операционная система вызывает организационный блок ОВ 80, если происходит одна из следующих ошибок:

- превышает время мониторинга цикла сканирования программы;
- ошибка при попытке доступа к организационному блоку, который еще обрабатывается или вызывается слишком часто в данном приоритетном классе);
- ошибка прерывания по времени суток (ошибка TOD-прерывания) - время, установленное для TOD-прерывания, прошло из-за перевода часов реального времени вперед или из-за более позднего перехода в режим RUN).

Если блок ОВ 80 недоступен в момент, когда произошла ошибка времени, то CPU переходит в режим STOP (СТОП). CPU также переходит в режим STOP, если организационный блок ОВ 80 вызывается второй раз в том же самом цикле сканирования программы.

Операционная система вызывает организационный блок ОВ 81, если происходит одна из следующих ошибок:

- если разряжена хоть одна резервная батарея в центральном контроллере или в блоке расширения;
- если не подается напряжение от батареи в центральном контроллере или в блоке расширения;
- если отказал источник питания на 24 В в центральном контроллере или в блоке расширения.

Организационный блок ОВ 81 вызывается для обработки как "приходящих" ("incoming"), так и для "исходящих" ("outgoing") событий. Если блок ОВ 81 недоступен в момент, когда произошла ошибка в блоке питания, CPU продолжает функционирование.

Операционная система выполняет мониторинг модульной конфигурации системы, проверяя ее один раз в секунду. Каждый раз, когда какой-либо модуль устанавливается в стойку или удаляется из стойки в режимах RUN, STOP или START-UP, соответствующая информация поступает в диагностический буфер и в список состояний системы.

Кроме того, при этом операционная система вызывает организационный блок ОВ 83, если CPU находится в режиме RUN. Если блок ОВ 83 недоступен в момент, когда произошло прерывание установки или удаления модуля, CPU переходит в режим STOP (СТОП). Может пройти максимум одна секунда, прежде чем будет возбуждено "прерывание установки/удаления модуля" после установки или удаления модуля. Вследствие чего за этот промежуток времени (между удалением модуля и генерацией соответствующего прерывания) может произойти ошибка доступа или ошибка, связанная с обновлением отображения процесса.

Если соответствующий модуль вставляется в сконфигурированный слот, CPU автоматически параметризует этот модуль, используя записи данных, ранее сохраненные в данном CPU. Организационный блок ОВ 83 вызывается тогда только для того, чтобы сигнализировать о том, что подключенный модуль готов к работе.

Операционная система вызывает организационный блок ОВ 84, если происходит или исчезает ошибка интерфейса (подсети MPI или PROFIBUS DP). Если блок ОВ 84 недоступен в момент, когда произошел отказ аппаратной части CPU, то CPU переходит в режим STOP.

Операционная система вызывает организационный блок ОВ 85, если происходит одна из следующих ошибок:

- если поступает запрос на запуск организационного блока, который не был загружен;
- ошибка при попытке доступа операционной системы к блоку (например, отсутствует экземплярный блок данных для вызываемого системного функционального блока SFB);
- ошибка доступа к периферии во время (автоматического) обновления образа процесса (со стороны системы).

В S7-400 CPU и CPU 318 организационный блок ОВ 85 вызывается при каждой ошибке доступа к периферии (I/O), например, во время обновления образа процесса в каждом цикле. При этом при каждом обновлении образа процесса в соответствующие байты в таблице входов образа процесса записываются либо нулевые значения, либо значения замены.

В S7-300 CPU (за исключением CPU 318) организационный блок ОВ 85 не вызывается в случае ошибки доступа к периферии (I/O) во время автоматического обновления образа процесса. При первой ошибке доступа к периферии в соответствующие байты в таблице входов образа процесса записываются либо нулевые значения, либо значения замены, после чего образ процесса больше не будет обновляться.

В соответствующим образом оснащенных CPU Вы можете использовать параметризацию CPU для установления режима вызова блока OB 85 для случаев ошибки доступа со стороны системы к периферии (I/O):

- организационный блок OB 85 будет вызываться при каждой ошибке доступа; в соответствующий входной байт записывается либо нулевое значение, либо значение замены;
- организационный блок OB 85 будет вызываться при первой ошибке доступа к периферии с атрибутом "incoming" ("приходящее событие"); при этом в соответствующие байты в таблице входов образа процесса будут записываться либо нулевые значения, либо значения замены, после чего образ процесса больше не обновляется; если ошибка устраняется, блок OB 85 вновь вызывается, но с атрибутом "outgoing" "уходящее событие", после чего образ процесса продолжает обновляться в обычном режиме;
- организационный блок OB 85 не будет вызываться в случае ошибки доступа; в соответствующий входной байт один раз будет записано либо нулевое значение, либо значение замены, после чего образ процесса больше не обновляется.

Если блок OB 85 недоступен в момент, когда произошла ошибка выполнения программы, то CPU переходит в режим STOP.

Операционная система вызывает организационный блок OB 86, если она обнаружила отказ стойки (авария в системе питания, обрыв провода, отказ интерфейсного модуля IM), отказ в подсети или в станции периферии. Организационный блок OB 86 вызывается и в случае "приходящего события", и в случае "уходящего события".

В мультипроцессорном режиме организационный блок OB 86 вызывается во всех CPU, если произошел отказ стойки.

Если блок OB 86 недоступен в момент, когда произошел отказ стойки, то CPU переходит в режим STOP.

Операционная система вызывает организационный блок OB 87, если происходит коммуникационная ошибка. Ниже представлены примеры некоторых коммуникационных ошибок:

- при использовании обмена посредством глобальных данных обнаружена неправильная идентификация фрейма или длина фрейма;
- невозможна передача диагностического сообщения;
- ошибка синхронизации часов;
- информация о состоянии (статусе) GD не может быть внесена в блок данных.

Если блок OB 87 недоступен в момент, когда произошла коммуникационная ошибка, то CPU переходит в режим STOP (СТОП).

Операционная система H CPU вызывает организационный блок OB 70, если происходят ошибки резервирования периферии - потеря резервирования в подсети PROFIBUS-DP, например, в случае отказа шины в активном ведущем (master) DP-устройстве или в случае отказа интерфейса ведомого (slave) DP-устройства.

Если блок OB 70 недоступен в момент, когда произошла ошибка резервирования периферии, CPU продолжает функционирование.

Операционная система H CPU вызывает организационный блок OB 72, если происходит одна из следующих ошибок:

- потеря резервирования CPU;
- ошибка сравнения (например, в RAM-памяти, в PIQ-области);
- переключение на резервное ведущее устройство;
- ошибка синхронизации;
- ошибка в подмодуле SYNC;
- прерывание обновления.

Если блок OB 72 недоступен в момент, когда произошла ошибка резервирования CPU, CPU продолжает функционирование.

5.10.3. Системная диагностика

Системная диагностика представляет из себя систему обнаружения, оценки и сообщения об ошибках, происходящих в программируемых контроллерах. Примеры ошибок: ошибки в пользовательской программе, отказы модулей, обрывы проводов в сигнальных модулях. Примеры *диагностических событий*:

- диагностические прерывания, поступающие от модулей, которые оснащены системой диагностики;
- сообщения о системных ошибках и переходах CPU из одного рабочего режима в другой;
- пользовательские сообщения, создаваемые посредством системных функций.

Модули со встроенной системой диагностики могут генерировать диагностические события двух видов: программируемые диагностические события и непрограммируемые диагностические события.

Программируемые диагностические события создаются только при условии, что пользователь устанавливает параметры, разрешающие работу системы диагностики, тогда как непрограммируемые диагностические события генерируются всегда и не зависят от параметров, разрешающих или запрещающих работу системы диагностики.

В случае поступления диагностического события:

- включаются светоиндикаторы (светодиоды), расположенные на передней панели CPU;
- сигнал о диагностическом событии передаётся в операционную систему CPU;
- генерируется диагностическое прерывание, если Вы установили параметры, разрешающие генерацию этого прерывания (по умолчанию диагностические прерывания блокируются).

Сигналы о всех диагностических событиях, передаваемые в операционную систему CPU, заносятся в *диагностический буфер (diagnostic buffer)* в том порядке, в котором они поступают, с фиксацией даты и времени поступления. Диагностический буфер - это область памяти в CPU с резервным питанием от батареи, которая сохраняет свое содержимое, даже когда выполняется сброс памяти. Диагностический буфер представляет из себя кольцевой буфер, размер которого определяется типом CPU. Когда диагностический буфер заполняется, то каждое вновь поступающее сообщение записывается в буфер поверх самого старого сообщения.

Пользователь может в любой момент считать содержимое диагностического буфера с помощью программирующего устройства PG. В блоке параметров *System Diagnostics* (системной диагностики) CPU Вы можете определить, желаете ли Вы расширить диапазон сообщений системной диагностики (включить сообщения о всех вызовах организационных блоков). Вы можете также активизировать опцию пересылки последнего диагностического сообщения в адрес определенного узла в MPI-подсети перед тем, как CPU перейдет в режим STOP.

Системная функция SFC 52 (WR_USMSG) позволяет записать сообщение в диагностический буфер, которое может быть послано в адрес всех узлов MPI-шины.

Запись в диагностическом буфере по формату соответствует записи системного события, например, стартовой информации организационного блока. В допустимых пределах Вы сами можете выбрать собственный идентификатор сообщения (записи) (параметр EVENTN) и дополнительную информацию (в параметрах INFO1 и INFO2). Идентификатор ID соответствует первым двум байтам записи в буфере.

Разрешенные для пользователя сообщения относятся к классам: 8 (диагностическая запись для сигнального модуля), 9 (стандартная запись пользователя), A и B (произвольные сообщения пользователя). Для дополнительной информации (INFO1) предназначены байты 7 и 8 (одно слово) и для дополнительной информации (INFO2) предназначены байты с 9 по 12 (двойное слово). Содержимое обеих переменных отдано на усмотрение пользователя.

Для того, чтобы переслать диагностическое сообщение в адрес соответствующего узла установите параметр SEND в состояние "1". Даже если пересылка сообщения невозможна (например, из-за того, что узел не указан, или из-за того, что буфер для пересылки (Send) переполнен) ввод данной записи будет сделан в диагностический буфер (если установлен бит 9 в идентификаторе события).

Если диагностическим событием является "приходящее" или "уходящее" диагностическое прерывание, то операционная система прерывает сканирование пользовательской программы и вызывает организационный блок OB 82. Если блок OB 82 не запрограммирован, то CPU переходит в режим STOP при возникновении диагностического прерывания. Выполнение организационного блока OB 82 может быть заблокировано или, наоборот, разрешено с помощью системных функций SFC 39 (DIS_IRT) и SFC 40 (EN_IRT) соответственно, а может быть заблокировано на время (отложено), а затем вновь разблокировано с помощью системных функций SFC 41 (DIS_AIRT) и SFC 42 (EN_AIRT) соответственно.

В первом байте стартовой информации B#16#39 устанавливается для "приходящего" диагностического прерывания, а B#16#38 устанавливается для "уходящего" диагностического прерывания. В шестом байте представлен идентификатор адреса (B#16#54 устанавливается для входов и B#16#55 - для выходов). Следующая переменная INT содержит адрес модуля, который генерирует диагностическое прерывание. В следующих четырех байтах содержится диагностическая информация, выдаваемая данным модулем.

В блоке OB 82 Вы можете использовать системную функцию SFC 59 (RD_REC) - "read data record" ("считать запись данных"), которая позволяет получить детальную информацию об ошибке. Диагностическая информация является консистентной до момента выхода из блока OB 82, то есть данные остаются "замороженными" ("frozen"). Выход из OB 82 подтверждает диагностическое прерывание в модуле.

Диагностические данные модуля содержатся в записях данных DS0 и DS1. Запись данных DS0 содержит четыре байта диагностических данных, описывающих текущее состояние модуля. Содержимое этих четырех байтов идентично содержимому байтов с 8 по 11 стартовой информации блока OB 82.

Запись данных DS1 содержит четыре байта из записи данных DS0 и, кроме того, диагностические данные, которые определяются типом модуля.

Список состояний системы (SSL - "system status list") описывает текущее состояние программируемого контроллера. С помощью "информационных" функций этот список может быть считан (но не может быть изменен). Вы можете считать часть списка (то есть "подсписок") с помощью системной функции **SFC 51 (RDSYSST)**. Подсписок - это "виртуальный" список, что означает, что такие списки могут быть представлены операционной системой CPU только по запросу.

Если параметр REQ = "1", то инициируется операция считывания. Параметр BUSY = "0" сообщает пользователю о том, что операция считывания уже завершилась. Операционная система может выполнить несколько асинхронных операций считывания "как бы одновременно". Ресурсы CPU могут использоваться многими "потребителями". Если функция SFC 51 сообщает пользователю о том, что недостаточно ресурсов (посредством значения функции равного W#16#8085), то Вы должны будете повторить запрос на считывание.

Содержимое параметров SZL_ID и INDEX зависит от CPU. Параметр SZL_HEADER имеет тип данных STRUCT, с переменными LENGTHDR (тип данных WORD) и N_DR (тип WORD) в качестве составляющих структуры. В параметре LENGTHDR содержится длина записи данных, а в параметре N_DR содержится число записей данных, которые должны быть считаны.

Параметр DR используется для определения переменной или области данных, в которые функция SFC 51 должна переслать считанные записи данных. Например, P#DB200.DBX0.0 WORD 256 описывает область размером в 256 слов данных в блоке данных DB 200, начиная с DBB 0.

Если предоставляемая область назначения имеет недостаточный размер, то в нее будет внесено максимально возможное количество записей данных. При этом данные переносятся только полными записями данных. Область назначения должна иметь размер, позволяющий вместить по крайней мере одну запись данных.

6. Обработка переменных

Данный раздел рассматривает вопросы обработки сложных переменных, для которых требуются в первую очередь знание структуры данных различных типов, умение использовать косвенную адресацию и способность определять адреса переменных во время выполнения программы.

Доступ к переменным простых типов данных (data types) возможен непосредственно с помощью STL-операторов, если Вы имеете дело с двоичными логическими операциями, с операциями с памятью или с операциями загрузки или пересылки. Для сложных и пользовательских типов данных прямой доступ в настоящее время возможен только к их отдельным компонентам. И если Вам все-таки необходимо организовать доступ к этим типам данных, то Вы должны знать внутреннюю структуру таких переменных.

Косвенная адресация (indirect addressing) предоставляет доступ к операндам, адреса которых заранее неизвестны (до момента выполнения программы). Пользователь может выбирать между косвенной адресацией посредством памяти (memory-indirect addressing) и косвенной адресацией посредством регистра (register-indirect addressing). Вы можете дождаться выполнения программы, для использования области операнда. Косвенная адресация позволяет Вам получать доступ к переменным сложных типов и пользовательских типов, используя абсолютную адресацию.

При использовании **прямого доступа к переменным** (direct variable access) используется текущий адрес локальной переменной. Если Вы определили адреса, Вы можете обработать соответствующие локальные переменные (в том числе и параметры блока), относящиеся к любому типу данных. В двух предыдущих главах содержится информация, требуемая для этих целей.

6.1. Типы данных

Тип данных определяет их свойства и характеристики, особое представление содержимого одного или больше связанных адресов и допустимых областей. STEP 7 предоставляет пользователю возможность предопределения данных, которые он может компилировать, как и определенные им пользовательские типы данных. Типы данных доступны везде. Они могут использоваться в любом блоке.

6.1.1. Простые типы данных

Переменные простых типов имеют максимальную длину, равную одному двойному слову; поэтому они могут быть обработаны с помощью функций загрузки и пересылки или с помощью двоичных логических операций.

Данные простого типа могут занимать один бит, один байт, одно слово и одно двойное слово.

Объявление (Declaration):

varname : datatype := pre-assignment,

где *varname* - имя переменной,
datatype - простой тип данных,
pre-assignment - (предопределенное) фиксированное значение переменной.

Идентификаторы типов данных (например, BOOL, REAL) являются ключевыми словами, они могут быть записаны также и в нижнем регистре. Переменная простого типа данных может быть объявлена в таблице символов глобальной или локальной переменной в разделе объявлений.

Переменная может быть предопределена во время объявления (в отличие от параметров функции или временных переменных). Значение предопределения должно относиться к тому же типу данных, что и сама переменная.

Вы можете применять переменные простых типов в качестве фактических параметров для соответствующим образом объявленных параметров блока (того же типа данных, POINTER или ANY) или Вы можете организовать к ним доступ посредством обычных STL-операторов (например, двоичная проверка, функции загрузки).

Переменные простых типов сохраняются таким же образом, как соответствующие адреса. Для этого допустимо использовать все адресные области, включая параметры блоков.

Переменная типа BOOL представляет собой значение бита (например, входа I 1.0). Переменные типов данных BYTE, WORD и DWORD представляют собой последовательности битов, состоящие соответственно из 8, 16 и 32 битов.

Специальной формой этих типов данных являются BCD-числа (числа в двоично-десятичном коде) и значения счетчиков, которые используются в сочетании с собственно функцией счетчика, а также тип данных CHAR, представляющий символ в коде ASCII.

В общем случае BCD-числа - это беззнаковые числа. При использовании в функции преобразования знак BCD-числа хранится в крайней левой (старшей) тетраде. Это ведет к потере одной тетрады BCD-числа для его диапазона.

В случае, если BCD-число хранится в шестнадцатиразрядном слове, знак хранится в старшей тетраде в бите 15. При этом, если состояние сигнала данного бита равно "0", то знак числа положительный, если состояние сигнала равно "1", то знак - отрицательный. Знак не влияет на значения отдельных тетрад. Аналогичное назначение применяется для 32-разрядного слова. Диапазон BCD-числа в этом случае составляет от 0 до 99999999.

BCD-числа не имеют специального идентификатора в STL. Вы можете вводить BCD-число с типом данных 16# (шестнадцатеричное), используя только цифры из ряда 0 - 9.

BCD-числа встречаются при загрузке значений таймеров и счетчиков и при работе с функциями преобразования. Тип данных S5TIME# может использоваться для задания значения таймера при запуске функции таймера, а для определения значения счетчика используется тип данных: 16# или C#. Значение счетчика с типом данных C# - это BCD-число из диапазона 000 - 999, в котором знак всегда равен 0.

Переменная типа CHAR (character - символ) занимает один байт. Тип данных CHAR представляет отдельный символ в формате ASCII. Например, символ 'A'. Вы можете также записать любой печатаемый символ в одинарных кавычках.

В операторах загрузки в языке программирования STL используются также некоторые специальные символы. Например, оператор L '\$\$' загружает знак доллара в коде ASCII. Кроме того, Вы можете использовать другие специальные формы типа данных CHAR, когда загружаете в аккумулятор символы в коде ASCII.

Например, оператор L 'a' загружает один символ (в данном случае символ a) с выравниванием вправо в аккумуляторе; оператор L 'aa' загружает два символа, а оператор L 'aaaa' соответственно загружает четыре символа.

Таблица 6.1. Специальные символы для переменной типа CHAR

Символ CHAR	Hex	Значение
\$\$	24 _{hex}	Символ доллара
\$'	27 _{hex}	Одинарная кавычка
\$L или \$I	0A _{hex}	Новая строка
\$P или \$p	0C _{hex}	Новая страница
\$R или \$r	0D _{hex}	Возврат каретки
\$T или \$t	09 _{hex}	Табулятор

Для REAL-чисел существует различие между числами, которые могут быть представлены с общей точностью ("нормированные" или "нормализованные" числа с плавающей запятой) и числами, которые представляются как числа с ограниченной точностью ("ненормированные" или "денормализованные" числа с плавающей запятой). Диапазон значений нормированных чисел с плавающей запятой составляет:

от $-3.402\,823 \cdot 10^{+38}$ до $-1.175\,494 \cdot 10^{-38}$
и от $+1.175\,494 \cdot 10^{-38}$ до $+3.402\,823 \cdot 10^{+38}$.

Диапазон значений ненормированных чисел с плавающей запятой составляет:

от $-1.175\,494 \cdot 10^{-38}$ до $-1.401\,298 \cdot 10^{-45}$
и от $+1.401\,298 \cdot 10^{-45}$ до $+1.175\,494 \cdot 10^{-38}$.

S7-300 CPU не могут выполнять вычисления с ненормированными числами с плавающей запятой. Битовая структура в этом случае такова, что ненормированное число с плавающей запятой представляется как ноль. Если результат вычисления попадает в диапазон ненормированных чисел с плавающей запятой, то значение этого результата представляется нулевым с установкой битов состояния OV и OS ("нарушение численного диапазона").

CPU выполняет вычисления с полной точностью представления чисел с плавающей запятой. Из-за ошибок округления при преобразовании результаты, отображаемые программатором, могут отличаться от теоретически точного представления результата. В таблице 6.2 показаны внутренние границы диапазона числа с плавающей запятой.

Таблица 6.2 Внутренние границы диапазона числа с плавающей запятой

Знак	Экспонента	Мантисса	Значение
0	255	не равна 0	Некорректное число с плавающей запятой (не число)
0	255	0	+ бесконечность
0	1 - 254	любое значение	Положительное нормированное число с плавающей запятой
0	0	не равна 0	Положительное ненормированное число с плавающей запятой
0	0	0	+0
1	0	0	-0
1	0	не равна 0	Отрицательное ненормированное число с плавающей запятой
1	1 - 254	любое значение	Отрицательное нормированное число с плавающей запятой
1	255	0	- бесконечность
1	255	не равна 0	Некорректное число с плавающей запятой (не число)

Переменные типа **S5TIME** используются для инициализации таймера из SIMATIC-функций для базовых языков программирования STL, LAD и FBD. Тип данных S5TIME занимает 16-разрядное слово со структурой "1+3 тетрады". Время задается в часах, минутах, секундах и миллисекундах. Преобразование числа S5TIME-формата во внутреннее представление выполняется системой STEP 7. Внутренне число представляется как BCD-число (число в двоично-десятичном коде) из диапазона от 000 до 999. Временная база (time base) может принимать следующие значения: 10 мс (0000), 100 мс (0001), 1 с (0010), 10 с (0011). Время определяется как результат произведения временной базы на значение таймера.

Переменная типа **DATE** хранится в машинном слове как беззнаковое число с фиксированной запятой. Содержание переменной соответствует числу дней, начиная с 01.01.1990 г. Представление числа содержит год, месяц и день, разделенные тире.

Примеры: DATE#1990-01-01 (= 0000_{hex})
D#2168-12-31 (= FF62_{hex})

Переменная типа **TIME** занимает одно двойное слово. Представление числа содержит спецификации дня (d), часов (h), минут (m), секунд (s) и миллисекунд (ms); при этом отдельные спецификации могут быть пропущены. Содержание переменной интерпретируется как число миллисекунд и сохраняется в 32-разрядном числе со знаком с фиксированной запятой.

Примеры: TIME#24d20h31m23s647ms (= 7FFF FFFF_{hex})
TIME#0ms (= 0000 0000_{hex})
T#-24d20h31m23s647ms (= 8000 0000_{hex})

"Десятичное" представление для данных типа TIME также возможно, например, TIME#2.25h или T#2.25h.

Переменная типа **TIME_OF_DAY** занимает одно двойное слово. Содержание переменной интерпретируется как число миллисекунд (ms), начиная с начала дня (0.00 часов) и сохраняется в 32-разрядном беззнаковом числе с фиксированной запятой. Спецификация числа состоит из часов, минут и секунд, разделенных двоеточием. В спецификации возможно указывать число миллисекунд с записью после числа секунд и отделением от этого числа точкой. Значение для миллисекунд может быть опущено.

Примеры: TIME_OF_DAY#00:00:00 (= 0000 0000_{hex})
TOD#23:59:59.999 (= 0526 5BFF_{hex})

6.1.2. Сложные типы данных

К сложным типам данных относятся такие данные, которые целиком не могут быть непосредственно обработаны с помощью STL-операторов. Система STEP 7 определяет следующие четыре сложных типа данных:

- DATA_AND_TIME - дата и время суток (в двоично-десятичном или в BCD-коде),
- STRING - символьная строка, в которой может быть до 254 символов,
- ARRAY - массив ("field" - "поле", комбинация переменных одного типа),
- STRUCT - структура (комбинация переменных разного типа).

Данные типов STRING, ARRAY и STRUCT должны быть заранее определены (предопределены) пользователем - для типа STRING задается длина символьной строки, а для типов ARRAY и STRUCT определяются состав и размеры данных-компонентов.

Вы можете объявить переменные сложных типов только в блоках глобальных данных и в экземплярных блоках данных как временные локальные данные или как параметры блока.

Данные типа **DATA_AND_TIME** включают в себе информацию о дате и времени суток. Вместо идентификатора DATA_AND_TIME Вы можете использовать сокращение DT.

DATA_AND_TIME или DT - ключевые слова; они могут быть записаны и в нижнем регистре.

Переменная может быть предопределена на этапе объявления (не как параметр блока в функции, как входной или выходной параметр в функциональном блоке или как временная переменная).

Значение предопределения должно относиться к тому же типу данных, что и сама переменная. Значение предопределения должно иметь следующий формат:

DATA_AND_TIME#год-месяц-день-часы:минуты:секунды.миллисекунды,
DT#год-месяц-день-часы:минуты:секунды.миллисекунды

Спецификация миллисекунд может быть опущена.

Переменные типа DT Вы можете применять в качестве соответствующим образом объявленных параметров блока (того же типа данных - DT или ANY); например, они могут быть скопированы с помощью системной функции SFC 20 (BLKMOV).

Переменные типа DATA_AND_TIME или DT занимают 8 байтов. Переменные начинаются на границе слов (начиная с байта с четным адресом). Все спецификации доступны в BCD-формате (в формате двоично-десятичного числа).

Таблица 6.3 Примеры объявления переменных типов DT и STRING

Имя	Тип данных	Начальное значение	Комментарий
Date1	DT	DT#1990-01-01-00:00:00	Минимальное значение переменной типа DT
Date2	DATA_AND_TIME	DT#2089-12-31-23:59:59.999	Максимальное значение переменной типа DT
First_name	STRING[10]	'Jack'	Переменная типа STRING, заняты 4 символа из 10
Last_name	STRING[7]	'Daniels'	Переменная типа STRING, заняты все 7 символов из

			7
NewLine	STRING[2]	'\$R\$L'	Переменная типа STRING занята спецсимволами
EmptyString	STRING[16]	''	Переменная типа STRING без заполнения

Данные типа **STRING** - символьная строка, в которой может быть до 254 символов, предваряемых байтами со значениями Max и. Тек

- параметр, определяющий максимальное (длину строки), которые могут быть записаны в данную переменную. Тек - параметр, определяющий действительное число символов в строке.

Таким образом, переменная STRING может содержать от 0 до 254 символов. Параметр Max при записи может быть опущен; при этом редактор будет использовать для такой переменной стандартную (максимально возможную) длину, равную 254 байта. При использовании функций (FC) редактор не допускает спецификацию параметра "Max", и он будет использовать для такой переменной стандартную длину, равную 254 байта. STRING - ключевое слово, оно может быть записано также в нижнем регистре.

Переменная может быть предопределена на этапе объявления (не как параметр блока в функции, или в функциональном блоке и не как временная переменная). Значение предопределения задается символами в ASCII-коде, заключенными в одинарные кавычки, или с предшествующим знаком доллара, в случае применения особых символов (см. табл. 6.3).

Если значение предопределения короче, чем объявленная максимальная длина, то неиспользованные позиции в поле переменной не занимают. При дальнейшей обработке такой STRING-переменной учитываются только символы, занимающие позиции в поле переменной. Как значение предопределения допускается "пустая строка".

Переменные типа STRING Вы можете применять в качестве параметров блоков типа STRING или ANY; например, они могут быть скопированы с помощью системной функции SFC 20. Для обработки этих переменных используются стандартные функциональные блоки - "IEC-функции".

Переменная типа STRING имеет максимальную длину 256 символов, из которых 254 байтов могут использоваться собственно для хранения данных. Переменные типа STRING начинаются на границе слов (начинаются с байта с четным адресом).

При применении STRING-переменных пользователь определяет для них максимальную длину. "Текущая" длина (то есть, фактически занимаемая строкой символов, равная числу "непустых" символов) определяется при предопределении STRING-переменной или при ее обработке. В первом байте переменной содержится значение максимальной (заданной при объявлении переменной) длины; во втором байте содержится значение текущей длины переменной. Начиная с 3-го по счету байта располагаются данные - строка символов в формате ASCII.

Данные типа **ARRAY** представляет собой массив ("поле"), содержащий в себе комбинацию фиксированного числа переменных одинакового типа с предельными индексами. Указанные пределы являются целыми числами типа INT из диапазона -32768 - +32767; значение максимального индекса массива должно быть больше или равно минимальному его значению. Массив может быть многомерным. При этом каждая размерность должна также определяться своими нижним и верхним значениями. Максимальное число размерностей для массива ARRAY составляет 6:

Пользователь может предопределить на этапе объявления отдельные элементы массива (в отличие от параметров функции, проходных параметров в функциональном блоке или временных переменных). Тип данных предопределения должен соответствовать типу, заявленному для переменной.

Нет необходимости определять все элементы массива; если число предопределяемых элементов меньше числа элементов массива, то только первые элементы получают значения. Число предопределяемых элементов не может быть больше числа элементов массива. Значения предопределения должны разделяться запятыми. Многократное назначение одного и того же значения может быть выполнено с помощью взятия значения в скобки и указания перед скобками коэффициента повторения (см. табл. 6.4).

Таблица 6.4 Примеры объявления переменных типа ARRAY

Имя	Тип данных	Начальное значение	Комментарий
Meas_Val	ARRAY[1..24]	0.4, 1.5, 11 (2.6, 3.0)	Переменная типа ARRAY с 24 элементами типа REAL
	REAL		
Time_Of_Day	ARRAY[-10..10]	21 (TOD#08:30:00)	Массив переменных TOD с 21 элементом
	TIME_OF_DAY		
Result	ARRAY[1..24,1..4]	96 (L#-0)	Двумерный массив с 96 элементами
	DINT		

Переменные типа ARRAY Вы можете применять в качестве параметров блоков того же типа, с такой же структурой или как параметры блоков типа ANY. Например, они могут быть скопированы с помощью системной функции SFC 20. Вы можете также определить отдельные элементы массива в качестве параметров блока типа ARRAY, если параметры блока относятся к тому же типу данных, что и эти элементы. Если отдельные элементы массива относятся к простым типам данных, то Вы можете обрабатывать их с помощью обычных STL-операторов.

Доступ к элементу массива обеспечивается по имени массива и индексу в квадратных скобках. Индекс имеет фиксированное значение в STL и не может быть изменен во время выполнения программы (не может быть переменной).

Массив может быть многомерным. Максимальное число размерностей для массива ARRAY составляет 6. Многомерный массив подобен одномерному. Во время объявления переменных диапазоны индексов по каждой размерности отделяются друг от друга запятыми и заключаются в квадратные скобки. Обеспечивая доступ к элементам многомерного массива в STL, Вы всегда должны указывать индексы всех размерностей массива.

Переменные типа ARRAY всегда начинаются на границе слов (т.е., начинаются с байта с четным адресом). Переменная типа ARRAY занимает в памяти область до следующей границы слова. Элементы массива типа BOOL начинаются с младшего бита; элементы массива типа BYTE и CHAR начинаются с расположенного справа байта. При этом элементы массива следуют один за другим по порядку.

В многомерных массивах элементы хранятся, выстроенными в цепочки (в порядке следования размерностей), начиная с первой размерности. Для элементов, имеющих длину 1 бит или 1 байт, новая размерность всегда начинается *со следующего байта*, а для элементов других типов новая размерность начинается *со следующего слова* (то есть, со следующего четного байта).

Тип данных **STRUCT** определяет структуру данных, состоящую из фиксированного числа компонентов, которые могут относиться к различным типам данных.

Пользователь может предопределить на этапе объявления отдельные компоненты структуры (в отличие от параметров функции, проходных параметров в функциональном блоке или временных переменных). Тип данных предопределения должен соответствовать типу, заявленному для переменной.

Таблица 6.5. Пример объявления переменной типа STRUCT

Имя	Тип данных	Начальное значение	Комментарий
MotCont	STRUCT		Переменная простой структуры с тремя компонентами
On	BOOL	FALSE (ЛОЖЬ)	Переменная MotCont.On типа BOOL
Off	BOOL	TRUE (ИСТИНА)	Переменная MotCont.Off типа BOOL
Delay	S5TIME	S5TIME#5s	Переменная MotCont.Delay типа S5TIME
	END_STRUCT		

Переменные типа STRUCT Вы можете целиком применять в качестве параметров блоков или как параметры блоков типа ANY. Например, Вы можете скопировать с помощью системной функции SFC 20 BLKMOV содержимое переменной типа STRUCT. Вы можете также определить отдельные компоненты структуры в качестве параметров блока типа STRUCT, если параметры блока относятся к тому же типу данных, что и эти компоненты.

Если отдельные компоненты структуры относятся к простым типам данных, то Вы можете обрабатывать их с помощью обычных STL-операторов. Доступ к компоненту структуры обеспечивается с использованием имени структуры и имени компонента, разделённых точкой.

Переменные типа STRUCT всегда начинаются на границе слов (то есть, начинаются с байта с четным адресом); отдельные компоненты располагаются в памяти в том порядке, в котором они были описаны при объявлении. Переменная типа STRUCT занимает в памяти область до следующей границы слова. Элементы массива типа BOOL начинаются с младшего бита; элементы массива типа BYTE и CHAR начинаются с расположенного справа байта. Компоненты других типов начинаются на границе слов.

Структура может быть *вложенной*. Вложенная структура - это такая структура, которая сама является компонентом другой структуры. Глубина вложения для структур может достигать шести вложений.

Пользовательский тип данных (UDT - "User-defined Data Type") определяет некоторую структуру данных (как комбинацию компонентов, которые могут относиться к различным типам данных), относящуюся к глобальным данным. Вы можете использовать пользовательский тип данных (UDT), если такая структура данных часто встречается в Вашей программе или Вы просто желаете дать имя структуре данных.

Вы можете создать структуру данных UDT либо при инкрементном редактировании программы, либо в текстовом редакторе, создавая исходный файл программы.

Данные UDT являются глобальными, то есть, будучи один раз объявленными, они могут быть использованы во всех блоках. Доступ к данным UDT может быть организован с использованием имени

(доступ по символу); для этого данным UDT Вы должны назначить имя в таблице символов (symbol table).

Если Вы хотите передать в переменную структуру данных, определенную в UDT, назначьте переменной при ее объявлении пользовательский тип данных нужной структуры, как это делается при назначении "обычных" типов данных. Доступ к данным типа UDT может быть организован как по абсолютному адресу (UDT0 - UDT65535), так и по имени (символьная адресация).

Вы можете также определить тип данных UDT целому блоку данных. При программировании блока данных Вы можете назначить требуемый тип UDT блоку как структуре данных.

Вы можете создавать данные пользовательского типа (UDT) или с помощью утилиты SIMATIC Manager, выбрав сначала объект *Blocks (Блоки)*, а затем - опции меню: *Insert -> S7 Block -> Data Type (Вставка -> S7 Block -> Тип данных)* или в редакторе, выбрав опции меню: *File -> New (Файл -> Создать)* и задав затем "UDTn" в строке "Имя объекта".

Двойной щелчок на объекте UDT в окне программы позволит открыть окно таблицы объявления данных, которая выглядит точно также, как таблица объявления блока данных. Структура UDT программируется точно также, как блок данных: с отдельными строками для имени (Name), типа (Type), начального значения (Initial value) и комментария (Comment). Единственное различие между таблицами заключается в том, что здесь невозможно переключение к обзору данных (data view). Начальные значения, которые Вы запрограммируете для структуры UDT, пересылаются в переменные при объявлении последних.

6.2. Косвенная адресация

Косвенная адресация дает Вам возможность доступа к адресам, которые неизвестны до начала выполнения программы. С помощью косвенной адресации Вы также можете выполнять повторяющиеся разделы, части программы, такие, например, как циклы, и при этом при каждом проходе цикла Вы можете назначать разные адреса данных.

Так как при косвенной адресации адреса не вычисляются до выполнения программы, существует опасность того, что области памяти будут непреднамеренно перезаписаны. *В этом случае поведение программируемого контроллера может быть непредсказуемым! Будьте крайне осторожны при использовании косвенной адресации!*

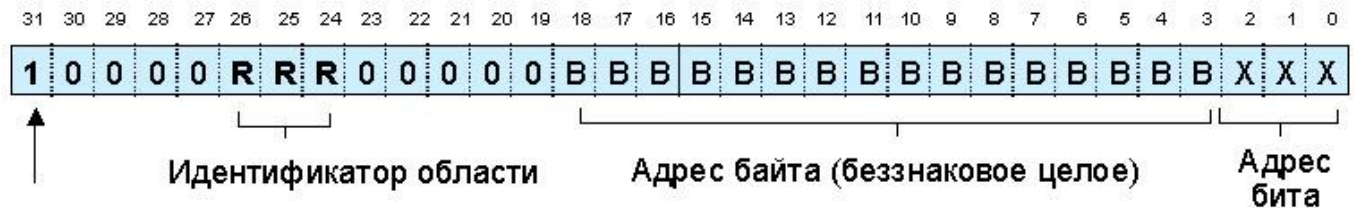
6.2.1. Указатели

Адрес при использовании косвенной адресации должен иметь такую структуру, в которой содержится адрес бита, адрес байта и, если необходимо, адрес области (*address area*). Поэтому такой адрес имеет специальный формат, имеющий название *указатель (Pointer)*. Указатель используется для "указания" на адрес.

Система STEP 7 различает три типа указателей:

- указатели на область (*area pointers*), длина такого указателя составляет 32 бита и содержит определенный адрес;
- указатели на DB (*DB pointers*), длина такого указателя составляет 48 битов и содержит кроме адреса (как в указателе на область) номер блока данных;
- ANY-указатели (*ANY pointers*), длина такого указателя составляет 80 битов и содержит кроме информации, общей с указателем на DB, например, тип данных операнда. Только первый тип указателей, т.е. указатели на область, имеют значение для косвенной адресации. Указатели на DB и ANY-указатели используются при передаче параметров блока. Так как эти типы указателей содержат указатели на область, в данном разделе описывается также и структура указателей на DB и ANY-указателей.

Указатели на область содержат адрес операнда и, возможно, также адрес области, где находится операнд. Если адрес (идентификатор) области не указывается, то это *внутризонный указатель (area-internal pointer)*, иначе, если адрес (идентификатор) области присутствует, то это *межзонный указатель (area-crossing pointer)*. Вы можете использовать указатель на область непосредственно и загружать в аккумулятор или в адресный регистр, так как длина такого указателя составляет 32 бита. Ниже представлена форма записи для представления константы для двух случаев: R#y.x - для случая внутризонного указателя, например, R#22.0 и R#Zy.x - для случая межзонного указателя, например, R#M22.0, где x - адрес бита, y - адрес байта, Z - "адрес области", в качестве которого используется идентификатор области. По значению бита 31 определяется тип указателя (внутризонный указатель или межзонный указатель) (см. рис. 6.1).



Bit 31=0: внутризонная
 Bit 31=1: межзонная

Рис. 6.1. Структура указателя на область.

Указатель на область содержит адрес бита, который всегда должен быть определен, даже для числовых операндов. Для числовых операндов адрес бита равен нулю. С помощью указателя на область R#M22.0 Вы можете, например, адресовать бит памяти M 22.0, а также байт в области памяти MB 22, слово MW 22 или двойное слово в области памяти MD22.

Указатели на DB содержат кроме адреса (как в указателе на область) номер блока данных в виде положительного целого числа типа INT. Этот номер идентифицирует блок данных, в то время как другая составная часть указателя содержит адрес в области глобальных данных или в области данных экземпляра. Во всех остальных случаях первые два байта указателя на DB будут содержать нуль.

Пример: R#DB 10.DBX 20.5
 R#DataBlock.DataAddress

Вы не можете загружать данный указатель в аккумулятор, тем не менее, Вы можете применять его в качестве параметра блока с типом POINTER для указания адреса данных. STEP 7 использует данный тип указателя для передачи фактических параметров.

ANY-указатели (ANY pointer) содержат кроме информации, общей с указателем на DB, код типа данных и коэффициент повторения. Это позволяет использовать такой указатель ANY для указания на область данных.

Указатель ANY используется в двух вариантах: для переменных с типом данных и для переменных с типом параметра. Если Вы используете указатель для переменных с типом данных, то указатель ANY содержит указатель на DB, тип и множитель повторения. Если Вы используете указатель для переменных с типом параметра, то указатель ANY содержит в дополнение к типу только номер вместо указателя на DB. Для функций таймера или счетчика тип повторяется в байте (n+6); байт (n+7) содержит B#16#00. Во всех остальных случаях эти два байта содержат значение W#16#0000.

Первый байт указателя ANY содержит синтаксическую структуру ID; в STEP 7 это всегда 10_{hex}. Код типа определяет тип данных переменных, для которых данный указатель ANY применяется. Переменные простых типов, DT и STRING принимают код, показанный в таблице 6.6, и размер 1.

Таблица. 6.6 Типы, используемые с ANY-указателем

Код типа	Тип	Код типа	Тип	Код типа	Тип
01	BOOL	08	REAL	17	BLOCK_FB
02	BYTE	09	DATE	18	BLOCK_FC
03	CHAR	0A	TOD	19	BLOCK_DB
04	WORD	0B	TIME	1A	BLOCK_SDB
05	INT	0C	S5TIME	1C	COUNTER
06	DWORD	0E	DT	1D	TIMER
07	DINT	13	STRING	00	NIL (Нуль-указатель)

Если Вы используете переменную типа массива ARRAY или структуры STRUCT (а также пользовательского типа UDT) в качестве параметра ANY, то редактор создает указатель ANY для массива или структуры. Этот указатель ANY будет содержать идентификатор для байта BYTE (02_{hex}) как тип и число байтов, определяя этим самым длину переменной как ее размер (quantity). Тип данных отдельного элемента массива или структуры при этом не имеют значения. Указатель ANY с двойным количеством байтов таким образом указывает на массив из слов (WORD).

Исключение: указатель на массив, состоящий из элементов типа CHAR также используется с типом CHAR (03_{hex}).

Вы можете применять указатель ANY в качестве параметра блока типа "параметр ANY", если необходимо указать на переменную или на адресную область.

Представление констант для типов данных выполняется следующим образом:
 R#[DataBlock.]Address Type Quantity (R#[БлокДанных.]Адрес Тип Размер).

Примеры:

- R#DB 11.DBX 30.0 INT 12 - область на 12 слов в блоке DB 11, начиная с DBB 30;
- R#M 16.0 BYTE 8 - область на 8 байт, начиная с MB 16;
- R#I 18.0 WORD 1 - слово входов IW 18;
- R#I 1.0 BOOL 1 - вход I 1.0.

Представление констант для *типов параметра* выполняется следующим образом:

L# Number Type Quantity (L# Номер Тип Размер).

Примеры:

- L# 10 TIMER 1 - функция таймера T 10;
- L# 2 COUNTER 1 - функция счетчика C 2.

Редактор использует указатель ANY, если тот согласован по типу и по размеру со спецификацией в представлении константы.

Необходимо отметить, что местоположение в памяти в указателе ANY для типов данных должно быть задано адресом бита.

Спецификация константы посредством указателя ANY имеет смысл, если Вы хотите получить доступ к области данных, для которой Вы не объявили переменной. В принципе, Вы можете также применить переменные или адреса в качестве параметра ANY. Например, представление "R#I 1.0 BOOL 1" идентично "I 1.0" или соответствующему символьному адресу.

С помощью типа параметра ANY Вы можете также объявить переменные во временных локальных данных. Вы можете использовать эти переменные для создания указателя ANY, который можно модифицировать во время выполнения программы.

Если Вы не выполняете никакого предопределения при объявлении параметра ANY в функциональном блоке, то редактор назначает значение 10_{hex} синтаксической структуре ID и 00_{hex} - всем остальным байтам. После этого редактор представляет такие (пустые) указатели ANY (при просмотре данных) следующим образом: R#P0.0 VOID 0.

6.2.2. Типы косвенной адресации в STL

Косвенная адресация возможна только с использованием абсолютных адресов. Вы не сможете применять косвенную адресацию с использованием символьных адресов (Вы должны также иметь возможность получать прямой доступ к отдельным элементам массива в STL). Если Вы желаете иметь косвенный доступ к переменной, Вы должны знать абсолютный адрес переменной. STL поддерживает прямой доступ к переменным.

Абсолютная адресация может применяться как:

- непосредственная адресация (immediate addressing);
- прямая адресация (direct addressing);
- косвенная адресация (indirect addressing).

Адресация посредством параметров блока - особая форма косвенной адресации: назначая фактические параметры параметрам блока, Вы определяете адреса, которые будут обработаны во время выполнения программы.

Мы имеем дело с *непосредственной адресацией*, когда численное значение задается непосредственно в операторе программы. Примером непосредственной адресации могут служить операция загрузки значения константы в аккумулятор, операция сдвига фиксированного значения, а также установка или сброс результата логической операции посредством операций SET (УСТАНОВКА) и CLR (ОЧИСТКА).

С помощью *прямой адресации* Вы получаете прямой доступ к адресу, например, A I 1.2 или L MW 122. Значение, с которым Вы хотите выполнить операцию, или значение, которое Вы хотите загрузить в аккумулятор, расположено по указываемому адресу, то есть в ячейке памяти. Вы адресуете данную ячейку памяти, указывая ее адрес непосредственно в операторе STL.

С помощью *косвенной адресации* STL-выражение указывает, где адрес может быть найден, вместо указания самого адреса. Мы различаем два типа косвенной адресации, различие между которыми определяется способом указания на адрес. Это:

- **косвенная адресация посредством памяти** (memory-indirect addressing), которая использует адрес в системной памяти для определения адреса; пример: в выражении T QW [MD 220] адрес слова выходов, в которые должна быть сделана пересылка, размещен в двойном слове меркеров MD 220;
- **косвенная адресация посредством регистра** (register-indirect addressing), которая использует адресный регистр для определения расположения адреса; пример: в выражении T QW [AR1,P#2.0] адрес слова выходов, в которые должна быть сделана пересылка, находится на 2 байта дальше адреса, размещенного в адресном регистре AR1.

Вы можете использовать косвенную адресацию посредством регистра в двух вариантах: как *внутризонную косвенную адресацию посредством регистра* (area-internal register-indirect addressing) и как

межзонную косвенную адресацию посредством регистра (*area-crossing register-indirect addressing*). С помощью внутризонной косвенной адресации посредством регистра Вы можете запрограммировать в операторе адресную область, адрес для которой указывается в адресном регистре. Следовательно, адрес в адресном регистре может быть изменен в пределах адресной области (например: выражение `L MW [AR1,P#0.0]` обеспечивает загрузку слова меркеров, адрес которого помещен в AR1).

С помощью межзонной косвенной адресации посредством регистра Вы определяете в выражении только длину (размер) области (бит, байт, слово, двойное слово). Адрес для адресной области указывается в адресном регистре, при этом он может изменяться динамически (например, выражение `L W [AR1,P#0.0]` обеспечивает загрузку слова, информация об адресной области и адрес которого находятся в AR1).

Адреса, которые могут быть определены косвенно, могут быть разделены на две категории:

- адреса, для которых может быть назначен простой тип данных;
- адреса, для которых может быть назначен тип параметра.

Вы можете использовать косвенную адресацию посредством памяти и косвенную адресацию посредством регистра с первой категорией из указанных адресов, но со второй категорией адресов Вы можете использовать только косвенную адресацию посредством памяти. Операнды, которые не могут иметь битовый адрес, также не требуют адреса бита в указателе, так что 16-разрядный номер вполне достаточен в качестве адреса (беззнаковое целое INT-число).

Область указателей имеет теоретический размер в пределах от 0 до 65535 (байтовых адресов или номеров). На практике верхняя граница для адресов в каждом случае зависит от CPU. Диапазон адресов для битов (битовых адресов) занимает значения от 0 до 7.

При косвенной адресации посредством памяти адрес указывается посредством адресованной ячейки памяти. Адрес должен иметь размер двойного слова, если требуется использовать указатель на область (*area pointer*), или же он должен иметь размер слова (WORD), если требуется при косвенной адресации использовать число в качестве указателя.

Адрес операнда может находиться в одной из ниже перечисленных адресных областей:

- область меркеров - как абсолютный адрес или символьная переменная;
- L-стек (область временных локальных данных) - как абсолютный адрес или символьная переменная;
- блок глобальных данных - как абсолютный адрес; при использовании адресации глобальных данных пользователь должен обеспечить, чтобы требуемый блок данных был открыт с помощью DB-регистра; если, например, Вы указываете адрес глобальных данных косвенным образом посредством двойного слова в глобальных данных, то все операции вычисления адреса должны быть произведены с учетом того, что все данные находятся в одном и том же блоке данных;
- экземплярный блок данных - как абсолютный адрес или символьная переменная; существуют определенные ограничения при использовании экземплярных данных в качестве адреса; (об этом см. ниже).

Если Вы используете экземплярные данные в качестве адресов в функциях, обрабатывайте их точно таким же образом, как и адреса глобальных данных; при этом Вы должны использовать DI-регистр вместо DB-регистра. Символьная адресация не допускается в этом случае. Вы можете также использовать экземплярные данные в качестве адресов в функциональных блоках, но только если Вы скомпилировали их как блоки `CODE_VERSION1` (без поддержки "мультиэкземплярности").

Указатель на область, используемый для косвенной адресации посредством памяти, всегда является внутризонным указателем. Это означает, что он содержит адрес байта и адрес бита. Если Вы хотите адресовать числовые операнды, то Вы должны всегда в качестве адреса бита указывать 0.

Пример: двойное слово MD 10 содержит указатель P#30.0. Выражение `A M [MD 10]` дает доступ к меркеру, адрес которого размещен в двойном слове MD 10; следовательно, с помощью этого выражения проверяется меркер M 30.0. С помощью выражения `L MW [MD 10]` Вы можете загрузить слово MW 30 в аккумулятор.

Вы можете использовать косвенную адресацию посредством памяти для всех бинарных операндов при использовании двоичных логических операций и операций с памятью, а также любых числовых операндов при использовании операций загрузки и передачи.

Пример. Косвенная адресация посредством памяти с указателем на область:

```
L P#30.0      //загрузка указателя в аккумулятор 1
T MD 10       //сохранение в двойном слове меркеров
L MW [MD 10]  //назначение дискретного адреса : исполнится L MW30
A M [MD 10]   //назначение двоичного адреса P#30.0: исполнится A M 30.0
```

Номер, используемый для косвенной адресации таймеров, счетчиков и блоков имеет длину 16 битов. Операнд, имеющий размер слова (WORD), достаточен для хранения номера.

Пример: слово MW 20 содержит число (номер) 133. Выражение `OPN DB [MW 20]` открывает блок глобальных данных, номер которого размещен в слове меркеров MW 20. С помощью выражения `SP T [MW 20]` Вы можете запустить таймер T 133 в режиме импульса.

Пример. Косвенная адресация посредством памяти по номеру:

```
L 133           //загрузка указателя в аккумулятор 1
T MW 20         //сохранение в слове меркеров
OPN DB [MW 20]  //открытие глобального блока DB : исполнится   OPN DB 133
SP T [MW 20]    //запуск таймера 133
```

С помощью косвенной адресации Вы можете использовать любые операции с таймерами и счетчиками. Вы можете открывать блоки данных или с использованием DB-регистра (OPN DB [*]), или с использованием DI-регистра (OPN DI [*]). Если слово адреса содержит 0, то CPU выполняет NOP-операцию.

Вы можете косвенно адресовать вызовы кодовых блоков, используя UC FC[*] и CC FC[*] или UC FB[*] и CC FB[*]. Вызов блоков посредством операций UC или CC просто приводит к смене блока; при этом не происходит ни передачи параметров блока, ни открытия экземплярного блока.

При косвенной внутризонной адресации посредством регистра адрес указывается посредством одного из двух адресных регистров. Содержимое адресного регистра является внутризонным указателем.

При косвенной адресации посредством регистра задается смещение в дополнение к информации адресного регистра. Это смещение добавляется к содержимому адресного регистра при выполнении соответствующей операции (без изменения содержимого адресного регистра). Указанное смещение имеет формат внутризонного указателя. Вы всегда должны задавать смещение при использовании данного вида адресации, и при этом Вы должны определять его константой. При косвенной адресации числовых операндов смещение всегда должно иметь нулевой битовый адрес. При этом максимальное значение указателя составляет не более P#8191.7.

Пример. Косвенная внутризонная адресация посредством регистра:

```
LAR1 P#10.0;    //загрузка указателя в адресный  регистр 1
L MW [AR1,P#4.0] //назначение дискретного адреса
A I [AR1,P#2.1]  //назначение двоичного адреса: исполнится   A I 12.1
Выражение A I [AR1,P#2.1] добавляет указатель к адресному регистру AR1 и, таким образом, формирует адрес входа, состояние которого должно быть проверено. С помощью выражения L MW [AR1,P#4.0] Вы можете загрузить слово меркеров MW 14 в аккумулятор 1.
```

Если адресный регистр содержит межзонный указатель, и если Вы используете данный адресный регистр для операций с внутризонной адресацией данных, то адрес области в адресном регистре будет игнорироваться.

Пример: Следующие инструкции обеспечивают загрузку в адресный регистр AR1 межзонного указателя на бит DBX 20.0 в глобальных данных, и после этого обеспечивается внутризонная адресация посредством адресного регистра AR1 в двойное слово меркеров. При выполнении следующего оператора выполняется загрузка в аккумулятор двойного слова MD 20.

```
LAR1 P#DBX 20.0
L MD[AR1,P#0.0]
```

При косвенной межзонной адресации посредством регистра адрес располагается в одном из двух адресных регистров. Содержимое адресного регистра является межзонным указателем (area-crossing pointer). При межзонной адресации Вы дополнительно устанавливаете адресную область посредством указателя на область (area pointer) в адресный регистр. Если Вы используете косвенную адресацию Вы задаете только идентификатор ID для спецификации длины адреса. Как и при внутризонной адресации Вы используете в данном случае смещение, которое Вы должны определить с помощью фиксированного значения для битового адреса. Содержание адресного регистра не изменяется при задании смещения.

Пример: Адресный регистр AR1 содержит указатель на область P#12.0 (при использовании инструкции LAR1 P#M12.0 Вы можете загрузить указатель непосредственно в адресный регистр AR1). Инструкция L B [AR1,P#4.0] добавляет указатель P#4.0 к адресному регистру AR1 и, таким образом, формирует адрес слова меркеров, которое должно быть загружено (MB 16 в данном случае). С помощью инструкции = [AR1,P#0.7] Вы можете назначить результат логической операции (т.е. RLO) биту M 12.7.

//Косвенная межзонная адресация посредством регистра

```
LAR1 P#M12.0    //загрузка указателя в адресный регистр 1
L B [AR1,P#4.0] //назначение дискретного адреса: исполнится   L M B16
...
= I [AR1,P#0.7] //назначение двоичного адреса
```

Вы не можете использовать межзонную адресацию для временных локальных данных (при этом CPU переходит в режим STOP). Вы должны использовать внутризонную адресацию, если адресуемая область расположена в области временных локальных данных.

Итак, в каких случаях использовать те или иные способы адресации? Если это возможно, то используйте косвенную внутризонную адресацию посредством регистра. Язык программирования STL более всего приспособлен для использования такой адресации. Вы можете видеть указание на адресную область, доступ к которой необходим для выполнения операции. И кроме того, CPU обрабатывает программу с косвенной внутризонной адресацией посредством регистра быстрее всего.

С другой стороны, косвенная адресация посредством памяти обеспечивает преимущество, если в выполняющуюся программу включены больше, чем два указателя. Однако необходимо учитывать период "валидности" (корректности) указателя: так, указатель в области меркеров может корректно использоваться без каких-либо ограничений при обработке программы целиком и даже на протяжении нескольких циклов сканирования программы; указатель в блоке данных может корректно использоваться пока блок находится в открытом состоянии; указатель в области временных локальных данных может корректно использоваться только во время выполнения блока.

Если адресные области должны быть доступны с использованием переменной адресации, то косвенная адресация посредством регистра является правильным выбором.

В таблице 6.7 представлены для сравнения варианты косвенной адресации. Все показанные фрагменты программ ведут к одному и тому же результату - к установке (в единицу) выхода Q 4.7.

Таблица 6.2 Варианты косвенной адресации

Косвенная адресация посредством памяти	Косвенная внутризонная адресация посредством регистра	Косвенная межзонная адресация посредством регистра
L P#4.7	LAR1 P#4.7	LAR1 P#Q4.7
T MD 24		
S Q [MD 24]	S Q [AR1, P#0.0]	S [AR1, P#0.0]

6.2.3. Использование адресных регистров

Ниже показаны инструкции, которые возможны на языке программирования STL в связи с использованием адресных регистров (рис. 6.2)

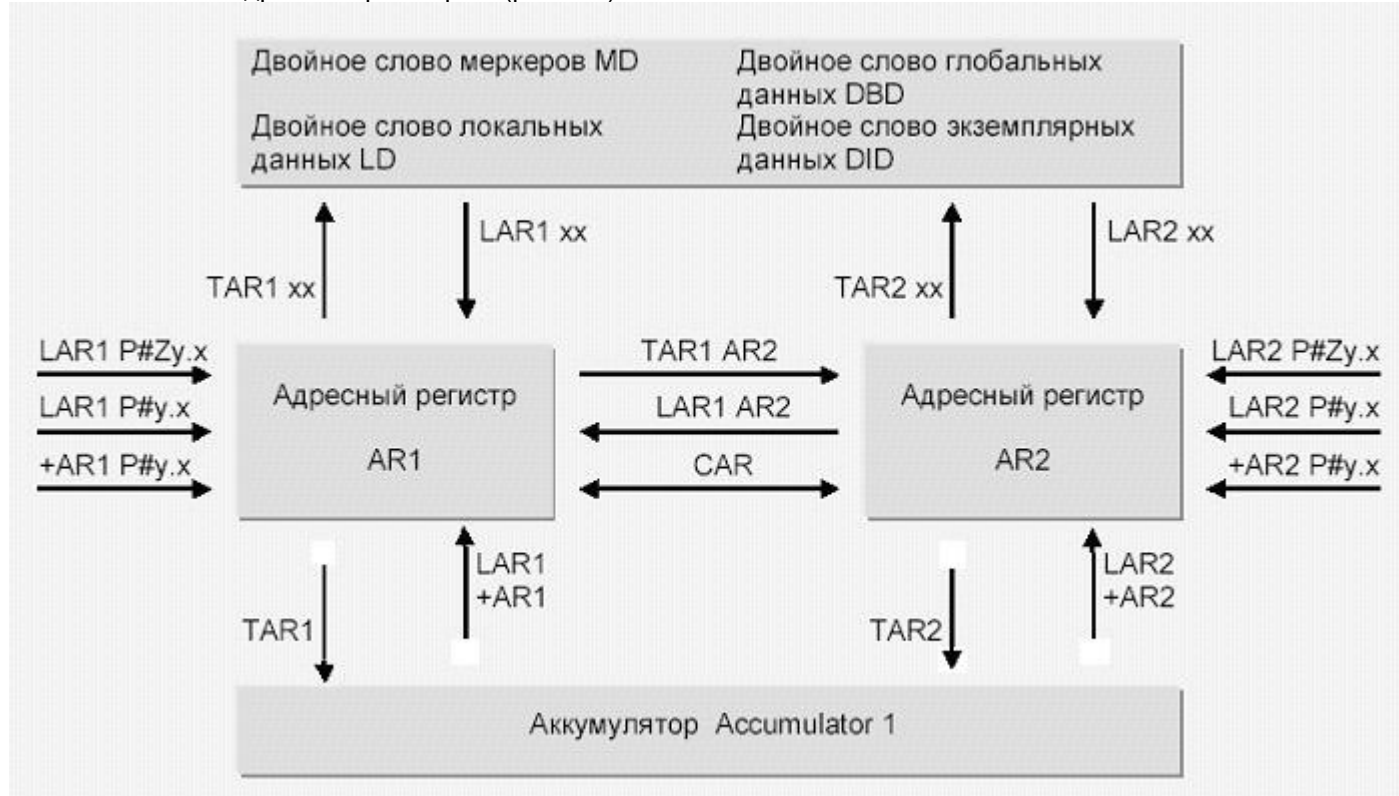


Рис. 6.2 Схема использования в STL адресных регистров

LARn - загрузка в адресный регистр ARn,
TARn - пересылка содержимого адресного регистра ARn,

+AR_n - сложение содержимое аккумулятора 1 и адресного регистра AR_n.

Оператор LAR_n загружает указатель на область в адресный регистр AR_n. Исходными данными для загрузки в адресный регистр Вы можете выбрать внутризонный или межзонный указатель или двойное слово из области меркеров, из области временных локальных данных, из области глобальных данных или из области экземплярных данных. При этом содержимое двойного слова должно соответствовать формату указателя на область.

В случае, если Вы не задаете адрес, оператор LAR_n загружает содержимое аккумулятора 1 в адресный регистр AR_n.

Если Вы используете инструкцию LAR1 AR2, то при выполнении данной инструкции происходит копирование содержимого адресного регистра AR2 в адресный регистр AR1.

Примеры:

```
LAR2 P#20.0      //Загрузка указателя P#20.0 в AR2
L P#24.0          //Загрузка указателя P#24.0 в аккумулятор 1
LAR1              //Загрузка содержимого аккумулятора 1 в AR1
LAR1 MD 120       //Загрузка содержимого MD 120 в AR1
LAR1 AR2          //Загрузка содержимого AR2 в AR1
```

Оператор TAR_n выполняет пересылку указателя на область целиком из адресного регистра AR_n. Как область назначения Вы можете выбрать двойное слово из области меркеров, из области временных локальных данных, из области глобальных данных или из области экземплярных данных.

В случае, если Вы не задаете адрес, оператор TAR_n пересылает содержимое адресного регистра AR_n в аккумулятор 1. При этом предыдущее содержимое аккумулятора 1 сдвигается в аккумулятор 2; предыдущее содержимое аккумулятора 2 теряется. Содержимое аккумуляторов 3 и 4 остается без изменения.

Если Вы используете инструкцию TAR1 AR2, то при выполнении данной инструкции происходит копирование содержимого адресного регистра AR1 в адресный регистр AR2.

Примеры:

```
TAR2 MD 140      //Пересылка содержимого AR2 в MD 140
TAR1              //Пересылка содержимого AR1 в аккумулятор 1
TAR1 AR2         //Пересылка содержимого AR1 в AR2
```

Оператор CAR выполняет обмен содержимым между адресными регистрами AR1 и AR2.

Пример: пусть 8 байтов данных пересылаются между областью меркеров (MB 100) и областью данных (DB 20.DBB 200). Направление пересылки указывается в меркере M 126.6. Если меркер M 126.6 имеет состояние "0", то адресные регистры обмениваются содержимым.

```
LAR1 P#M100.0
LAR2 P#DBX200.0
OPN DB 20
A M 126.6
JC OV
CAR
OV: L D[AR1,P#0.0]
T D[AR2,P#0.0]
L D[AR1,P#4.0]
T D[AR2,P#4.0]
```

Пользователь может запрограммировать операцию сложения некоторого значения с содержимым адресного регистра для того, чтобы, например, инкрементировать адрес при каждом следующем проходе цикла в программе. Вы можете задать прибавляемое значение или как константу (как внутризонный указатель), или это значение может быть расположено в правом слове аккумулятора 1. Тип указателя в адресном регистре (внутризонный указатель или межзонный указатель) и адресной области при этом сохраняются.

Инструкции +AR1 P#y.x и +AR2 P#y.x прибавляют указатель к содержимому указанного адресного регистра AR1 или AR2. Необходимо отметить, что в данных инструкциях указатель на область может иметь максимальную величину P#4095.7. Если аккумулятор содержит величину, большую, чем P#4095.7, то число интерпретируется как инвертированное число с фиксированной запятой и, таким образом, вычитается.

Пример: пусть необходимо сравнить значение в машинном слове в области данных с некоторой величиной. Если сравниваемая величина больше, чем значение в области данных, то устанавливается меркер (получает значение "1"), в противном случае меркер сбрасывается (получает значение "0").

```
OPN DB 14
LAR1 P#DBX20.0
```

```

LAR2 P#M10.0
L Quantity_Data
Loop: T LoopCounter
L ComparisonVal
L W[AR1,P#0.0]
>I
= [AR2,P#0.0]
+AR1 P#2.0
+AR2 P#0.1
L LoopCounter
LOOP Loop

```

Инструкции +AR1 и +AR2 интерпретируют значение в аккумуляторе 1 как целое число формата INT, представляют его с соответствующим знаком как 24-разрядное число и прибавляют к содержимому указанного адресного регистра AR1 или AR2 как показано на Рис. 6.3. При отрицательном числе в аккумуляторе указатель будет уменьшен. Нарушение верхней границы допустимого диапазона байтового адреса (65535) не имеет значения: "избыточные" биты отрезаются.

Аккумулятор Accumulator 1



Адресный регистр

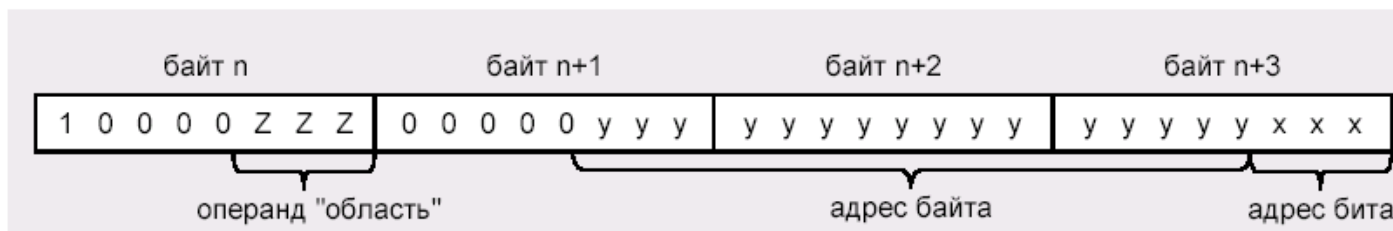


Рис. 6.3. Прибавление значения к адресному регистру

Необходимо отметить, что битовый адрес размещается в битах с номерами от 0 до 2. Если необходимо инкрементировать байтовый адрес в аккумуляторе 1, то увеличение надо начинать с бита 3 (то есть, прибавляемое значение Вы должны сдвинуть на три разряда влево).

Пример: Пусть в блоке данных DB 14 16 байтов, адреса которых вычисляются с помощью указателя в двойном слове меркеров MD 220 и смещения в байте меркеров MB 18, должны быть очищены. Перед прибавлением к AR1 содержимое MB 18 должно быть смещено вправо на 3 разряда (SLW 3).

```

OPN DB 14;
LAR1 MD 220;
L MB 18;
SLW 3;
+AR1 ;
L 0;
T DBD[AR1,P#0.0];
T DBD[AR1,P#4.0];
T DBD[AR1,P#8.0];

```

T DBD[AR1,P#12.0];

Примечание:

Системная функция SFC 21 FILL используется для заполнения больших областей определенным значением бита.

6.2.4. Особенности косвенной адресации

В языке программирования STL адресный регистр AR1 используется для обращения к параметрам блока, которые пересылаются как DB-указатели. При использовании функций адресный регистр AR1 может применяться для обращения ко всем параметрам блока сложных типов данных, а в случае функциональных блоков это касается входных/выходных параметров сложных типов. При использовании доступа к параметрам блока этого вида, например, для того, чтобы проверить битовый компонент структуры или записать целое INT-значение в элемент массива, содержимое адресного регистра AR1 изменяется и изменяется также содержимое DB-регистра. Это происходит тогда, когда Вы передаете параметры блока данного типа в вызываемые блоки.

Если Вы используете адресный регистр AR1 (помимо косвенной адресации), то при этом следует запретить доступ к параметру блока (в соответствии с изложенным выше материалом). Другими словами, Вы должны сохранять содержимое адресного регистра AR1 до обращения к параметрам и вновь загружать регистр после выполнения операции доступа.

Пример: Вы загружаете указатель в AR1 и используете этот адресный регистр для косвенной адресации. В определенный момент Вы хотите загрузить значение компонента структуры *Motor.Act*. Перед загрузкой *Motor.Act* Вы сохраняете содержимое DB-регистра и адресного регистра AR1; после выполнения операции загрузки вышеназванного компонента структуры, Вы восстанавливаете состояния регистров (см. текст программы).

```
...
//***** Сохранение адресного регистра AR1 *****
// AR1_Memory: DWORD
// DB_Memory: WORD
...
//Косвенная адресация с регистром AR1 и DB-регистром
LAR1 P#y.x
OPN DB z
...
//Сохранение содержимого регистров
L DBNO
T DB_Memory
TAR1 AR1_Memory
//Обращение к параметрам блока сложного типа
L Motor.Act
//Восстановление содержимого регистров
OPN DB [DBMemory]
LAR1 AR1_Memory
T DBW[AR1,P#0.0] //Сохранение загруженного значения
```

При использовании функциональных блоков с возможностью работы в "мультиэкземплярном режиме" (блок версии 2), адресный регистр AR2 используется в STEP 7 как "базовый адресный регистр" ("Base address register") для экземплярных данных. При вызове экземпляра адресный регистр AR2 содержит P#DBX0.0 и при всех обращениях к параметрам блоков или к статическим локальным данным в FB используется косвенная внутризонная адресация области DI посредством данного регистра.

При вызове локального экземпляра производится инкрементирование "базового адреса" при помощи +AR2 P#y.x, так что обращение к данным может быть выполнено относительно данного адреса внутри вызываемого функционального блока, который использует экземплярный блок данных вызывающего функционального блока. Таким образом, функциональные блоки могут быть вызваны и как автономные экземпляры, и как локальные экземпляры (к тому же в любой точке в функциональном блоке и даже несколько раз).

Если Вы программируете функциональный блок как блок версии 1, то есть, без "мультиэкземплярного режима", то STEP 7 не использует адресный регистр AR2.

Итак, Вы хотите использовать адресный регистр AR2 в функциональном блоке с возможностью работы в "мультиэкземплярном режиме", Вы должны сначала сохранить содержимое адресного

регистра AR2 и DI-регистра до использования и вновь восстанавливать данные в регистрах после выполнения операции доступа. Вы не должны программировать никаких обращений к параметрам блока или к статическим локальным данным в области, в которой Вы используете адресный регистр AR2. При использовании адресного регистра AR2 внутри функций никаких ограничений не накладывается.

Пример: в функциональном блоке Вам необходимо использовать регистр AR2 и DI-регистр для выполнения косвенной адресации. Прежде всего Вы должны сохранить содержимое DI-регистра и адресного регистра AR2. При этом Вы не должны выполнять обращения к параметрам блоков или к статическим локальным данным до того, пока Вы не восстановите содержимое регистров (см. текст программы).

```
...
//***** Сохранение адресного регистра AR2 *****
// AR2_Memory: DWORD
// DI_Memory: WORD
...
//Сохранение содержимого регистров
L   DINO
T   DI_Memory
TAR2 AR2_Memory
//Косвенная адресация с регистром AR2 и DI-регистром
LAR2 P#y.x
OPN DI z
...
L   DIW[AR2,P#0.0]
...
//Восстановление содержимого регистров
OPN DI [DI_Memory]
LAR2 AR2_Memory
```

При использовании функциональных блоков, скомпилированных с CODE_VERSION1 (без "мультиэкземплярного режима"), Вы можете использовать все операторы (инструкции), описанные в данной главе без ограничений.

При использовании функциональных блоков с возможностью работы в "мультиэкземплярном режиме", редактор обращается к экземплярным данным посредством адресного регистра AR2; то есть, все эти операции доступа носят косвенный характер. Это же касается косвенной адресации и обработки адресных регистров. Если Вы используете абсолютную адресацию для экземплярных данных, в которых Вы храните указатели на область, редактор использует абсолютные адреса. Тем не менее, как только Вы начнете использовать символьную адресацию, редактор отбросит эту часть программы как попытку "двойной косвенной адресации" (double indirect addressing).

В таблице 6.8 представлены два таких примера: если Вы используете косвенную адресацию посредством памяти в функциональных блоках с возможностью работы в "мультиэкземплярном режиме", то Вы не можете непосредственно использовать указатель, который Вы желаете сохранять в статических локальных данных. Вы должны скопировать указатель во временные локальные данные, и после этого можете работать с ним. При этом Вы не можете непосредственно загрузить указатель на статические локальные данные в адресный регистр и не можете также переслать содержимое адресного регистра непосредственно в указатель (второй пример).

Таблица 6.8. Различие в программировании FB версий 1 и 2

Для FB, скомпилированных с параметром CODE_VERSION1 (без "мультиэкземплярного режима")	Для FB, скомпилированных с параметром CODE_VERSION2 (с "мультиэкземплярным режимом")
// sPointer : DWORD	// sPointer : DWORD // tPointer : DWORD
L MW[sPointer]	L sPointer T tPointer L MW[tPointer]
LAR1 sPointer	L sPointer LAR1
TAR1 sPointer	TAR1 T sPointer

6.3. Прямой доступ к переменным

В данном разделе рассматривается, как реализовать прямое обращение к локальным переменным с использованием их абсолютных адресов. "Обычные" STL-операторы позволяют обращаться к локальным переменным простых типов. Локальные переменные сложных типов, а также параметры блоков типа POINTER или ANY не могут быть обработаны "целиком". Для обработки таких переменных Вы сначала должны вычислить начальный адрес, по которому переменная хранится, а затем Вы можете обращаться к отдельным ее частям, используя косвенную адресацию. Таким же путем Вы можете также обрабатывать параметры блоков, относящиеся к сложным типам данных.

6.3.1. Загрузка адреса переменной

Следующие инструкции дают начальный адрес локальной переменной:

L P#name

LAR1 P#name

LAR2 P#name

Здесь операнд *name* - это имя локальной переменной. В этих инструкциях межзонный указатель загружается в аккумулятор 1, в адресный регистр AR1 и в адресный регистр AR2 соответственно. Межзонный указатель содержит адрес первого байта переменной. Если имя *name* не может быть идентифицировано однозначно как локальная переменная, тогда добавьте символ "#" перед именем, так что инструкция изменится, например, следующим образом: L P##name /

В таблице 6.9 представлены области, в которых, в зависимости от вида блока, может быть размещена переменная *name*.

Таблица 6.9. Разрешенные области для размещения адресов переменных

Инструкция	Тип данных переменной <i>name</i>	Блок			
		ОВ	FC	FB V1	FB V2
L P#name	Временные локальные данные	x	x	x	x
	Статические локальные данные	-	-	x	x ¹⁾
	Параметр блока	-	x	x	x ¹⁾
LARn P#name	Временные локальные данные	x	x	x	x
	Статические локальные данные	-	-	x	x ¹⁾
	Параметр блока	-	-	x	x ¹⁾

¹⁾ Адрес переменной, связанный с адресным регистром AR2

В функциях FC адрес параметра блока не может быть загружен непосредственно в адресный регистр. Поэтому здесь следует использовать аккумулятор 1, например:

L P#name

LAR1

В функциональных блоках (FB), скомпилированных с ключевым словом CODE_VERSION1 (то есть, как блок версии 1, без "мультиэкземплярного режима"), абсолютный адрес переменной экземпляра может быть загружен.

В функциональных блоках (FB), скомпилированных как блок версии 2, то есть, с возможностью "мультиэкземплярного режима", абсолютный адрес (относительно адресного регистра AR2) переменной экземпляра может быть загружен, если переменная относится к статическим локальным данным или является параметром блока. Если необходимо вычислить абсолютный адрес переменной в экземплярном блоке данных, Вы должны сложить внутризонный указатель (только адрес) из AR2 с загруженным адресом переменной.

Пример 1: загрузить адрес переменной в адресный регистр AR1.

TAR2

L DW#16#00FF_FFFF // маска для выделения адреса

AD

LAR1 P#name

+AR1

Здесь адрес из AR2 загружается в аккумулятор 1 и добавляется к содержимому AR1 с помощью оператора +AR1. В результате чего адрес переменной *#name* загружается в адресный регистр AR1.

Пример 2: загрузить адрес переменной в аккумулятор 1.

TAR2

```

L DW#16#00FF_FFFF // маска для выделения адреса
AD
L P#name
+D

```

В данном примере в результате выполнения данного фрагмента программы в аккумуляторе 1 оказывается адрес переменной *#name*.

Сложение с внутренним указателем может быть пропущено, если указатель имеет значение P#0.0. Это для случая, если Вы не используете функциональный блок как локальный экземпляр.

Необходимо отметить, что при выполнении инструкции "LAR2 P#name" перезаписывается содержимое адресного регистра AR2, который применяется при использовании функциональных блоков с возможностью "мультиэкземплярного режима" как "базовый адресный регистр" при адресации экземплярных данных! С помощью этих (LARn) операторов загрузки Вы можете организовать доступ только к одной переменной целиком, но не можете организовать доступ к отдельным компонентам массивов, структур или локальных экземпляров. Вы не можете с помощью этих операторов загрузки получить доступ к переменным в блоках глобальных данных или в адресных областях входов, выходов, периферийных входов/выходов и в адресных областях меркеров.

В таблице 6.10 показано, как рассчитываются адреса переменных типов INT и STRING в статических локальных данных и как использовать эти адреса. Если Вы используете образец программы в функциональном блоке, который Вы вызываете как локальный экземпляр, Вы должны прибавить базовый адрес к адресу переменной, как показано выше.

Таблица 6.10. Загрузка адреса переменной (примеры)

//Объявление переменных (функциональный блок не является локальным экземпляром!)	
//Назначение переменных начинается с адреса P#0.0	
Field : ARRAY[1..22] OF BYTE	//переменная ARRAY занимает 22 байта
Number : INT := 123	//переменная INT занимает 2 байта
FirstName : STRING := 'Joane'	//переменная STRING занимает 5 байтов
LAR1 P#Number	Загрузка начального адреса <i>Number</i> в AR1. Регистр AR1 теперь содержит P#DIX22.0
L W[AR1,P#0.0]	Соответствует выражению L DIW 22 или L <i>Number</i>
LAR1 P#FirstName	Загрузка начального адреса <i>FirstName</i> в AR1. Регистр AR1 теперь содержит P#DIX24.0
L B[AR1,P#0.0]	Загрузка 1-го байта (максимальная длина строки символов) в аккумулятор 1
L B[AR1,P#2.0]	Загрузка 3-го байта (1-го байта данных) в аккумулятор 1
L 'John'	Запись имени 'John' в строку символов
T D[AR1,P#2.0]	
L 4	Корректировка текущего размера строки символов (=4)
T B[AR1,P#1.0]	Переменная <i>FirstName</i> теперь содержит 'John'

6.3.2. Хранение переменных

Редактор хранит отдельные переменные в блоках данных в порядке их объявления. При этом должны соблюдаться следующие правила.

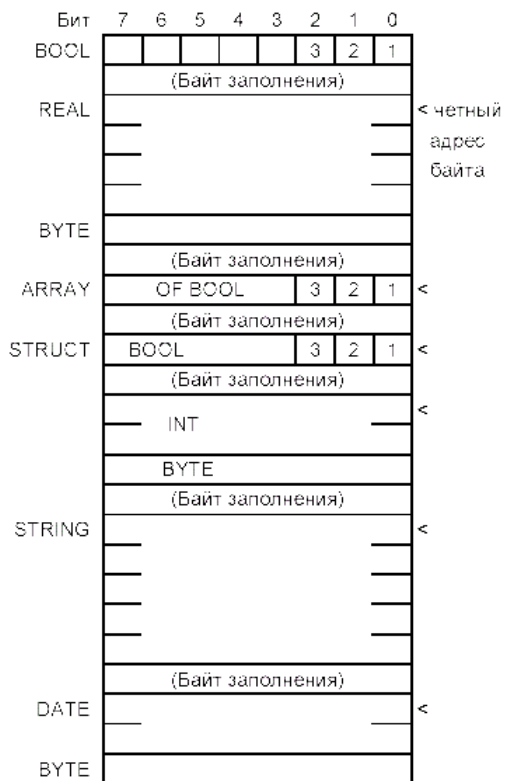
- Первая "битовая" переменной (с длиной в 1 бит) из непрерывной последовательности объявленных переменных размещается в бите 0 следующего байта; за первой битовой переменной размещаются следующие битовые переменные.
- "Байтовая" переменная (с длиной в 1 байт) из непрерывной последовательности объявленных переменных размещается в следующем байте.
- Переменные, имеющие размер одного "машинного" слова и имеющие размер двойного слова, всегда размещаются, начиная с границы слова, то есть в байте с четным адресом.
- Переменные типов DT и STRING также всегда размещаются, начиная с границы слова.
- Переменные типа ARRAY также всегда размещаются, начиная с границы слова, и заполняют область до следующей границы слова. Это касается также битовых и байтовых массивов. Элементы массивов простых типов сохраняются, как описано выше. Элементы массивов сложных типов размещаются, начиная с границы слова. Каждая размерность массива размещается с выравниванием как автономный массив.
- Переменные типа STRUCT также всегда размещаются, начиная с границы слова, и заполняют область до следующей границы слова. Это касается также "чисто" битовых и байтовых структур. Компоненты структур простых типов сохраняются, как описано выше. Элементы структур сложных типов размещаются, начиная с границы слова.

Группируя отдельные битовые переменные и компоуя байтовые переменные, Вы можете оптимизировать размещение данных в блоках данных.

На рисунке 6.4 Вы можете видеть один пример неоптимального и один пример оптимального хранения данных.

Неоптимальное размещение данных

```
DATA_BLOCK StorageNonOptimized
STRUCT
  Bit1  : BOOL
  Bit2  : BOOL
  Bit3  : BOOL
  Reall  : REAL
  Bytel  : BYTE
  Bit_field : ARRAY [1..3] OF BOOL
  Structure : STRUCT
    S_Bit1 : BOOL
    S_Bit2 : BOOL
    S_Bit3 : BOOL
    S_Intl : INT
    S_Byte : BYTE
  END_STRUCT
  Character : STRING[3];
  Datel  : DATE;
  Byte2  : BYTE;
END_STRUCT
```



Оптимальное размещение данных

```
DATA_BLOCK StorageNonOptimized
STRUCT
  Bit1  : BOOL
  Bit2  : BOOL
  Bit3  : BOOL
  Bytel  : BYTE
  Reall  : REAL
  Bit_field : ARRAY [1..3] OF BOOL
  Structure : STRUCT
    S_Bit1 : BOOL
    S_Bit2 : BOOL
    S_Bit3 : BOOL
    S_Byte : BYTE
    S_Intl : INT
  END_STRUCT
  Character : STRING[3]
  Byte2  : BYTE
  Datel  : DATE
END_STRUCT
```

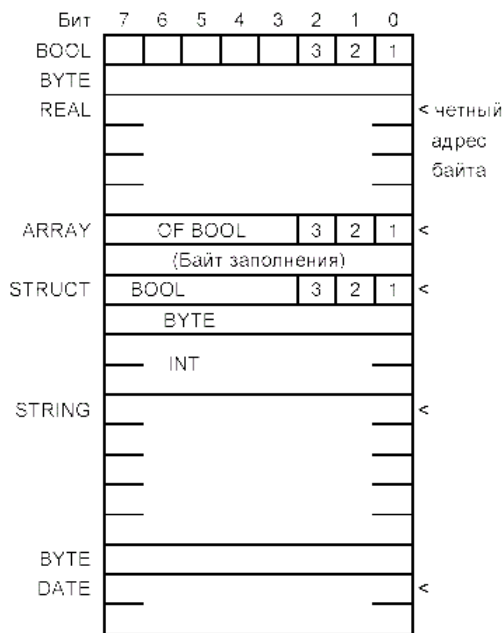


Рис. 6.4. Примеры размещения данных в блоке данных

Необходимо отметить, что редактор всегда "заполняет" переменные типа ARRAY и типа STRUCT до следующей границы слова, что означает, что ни битовые, ни байтовые переменные не могут быть размещены для хранения в остающихся незанятых областях памяти. Тем не менее, Вы можете оптимизировать размещение переменных внутри структуры.

6.3.3. Доступ к параметрам в функциях

Редактор сохраняет данные о параметрах блоков для функции (имеются в виду параметры функции, относящиеся к типу "параметр блока"; далее по тексту - "параметры функции") в виде межзонных

указателей области в соответствующем операторе вызова; таким образом, каждый параметр функции требует для хранения одно двойное слово памяти. В зависимости от типа данных и от объявленного типа указатель ссылается на фактический параметр, на копию фактического параметра во временных локальных данных вызывающего блока (созданную редактором) или на указатель во временных локальных данных вызывающего блока, который в свою очередь указывает на фактический параметр (см. табл. 6.11). Исключение: для следующих типов параметров: TIMER, COUNTER и BLOCK_хх указателем является 16-разрядное число, расположенное в левом слове параметра блока (функции).

Таблица 6.11. Доступ к параметрам функции

Тип данных	INPUT	IN_OUT	OUTPUT
	Параметр является указателем на область для:		
Простой тип	значения	значения	значения
Сложный тип	DB-указателя	DB-указателя	DB-указателя
TIMER, COUNTER и BLOCK	номера	недопустимый	недопустимый
POINTER	DB-указателя	DB-указателя	DB-указателя
ANY	ANY-указателя	ANY-указателя	ANY-указателя

В случае простых типов данных параметр блока указывает непосредственно на фактический параметр. Тем не менее, с помощью указателя на область как параметр блока Вы не можете получить доступ ни к каким константам или к операндам, расположенным в блоках данных. По этой причине еще на этапе компилирования редактор копирует константы или фактические параметры, размещенные в блоке данных (операнд с полным адресом), в область временных локальных данных вызывающего блока и "направляет" указатель на эти копии данных. Подобная область параметров называется "V" (временные локальные данные "предыдущего" блока или V-область).

Перед текущим вызовом функции (FC) имеет место копирование в V-область данных, полученных при вызове функции: входных параметров и входных-выходных параметров, а после вызова функции имеет место копирование возвращенных входных-выходных параметров, выходных параметров, а также значения функции. Поэтому здесь действует правило, согласно которому Вы можете только проверить входные параметры и сделать запись в выходные параметры. Если Вы передаете значение во входной параметр, используемый в операнде с полным адресом, то это значение будет сохранено во временных локальных данных вызывающего блока и будет в дальнейшем "забыто" (утеряно), так как в дальнейшем не будет никакого "обратного" копирования в "фактическую" ("actual") переменную в блоке данных. Похожая история и с загрузкой соответствующего выходного параметра: так как никакого копирования не происходит из фактической переменной из блока данных в V-область, Вы сможете загрузить лишь "неопределенное" значение из V-области.

Из-за операции копирования выходной параметр, как и значение функции, **должен** быть перезаписан определенным значением при простом типе данных в блоке, если операнд с полным адресом рассматривается или может рассматриваться как фактический параметр. Если Вы не назначаете значения для выходного параметра (например, из-за преждевременного выхода из блока или из-за обхода в программе соответствующей инструкции), значение во временных локальных данных также не будет инициализировано. Оно будет иметь значение, которое было перед вызовом блока. В дальнейшем выходной параметр будет перезаписан этим "неопределенным" значением.

В случае сложных типов данных, таких как DT, STRING, ARRAY, STRUCT, а также UDT, фактические параметры размещаются или в блоке данных, или в V-области. Так как указатель на область не может обеспечить доступ к фактическим параметрам в блоке данных, то редактор создает DB-указатель в V-области во время процесса компиляции. Этот DB-указатель указывает на фактический параметр в блоке данных (если номер DB не равен 0), или указывает на V-область (если номер DB равен 0). DB-указатели для всех объявленных типов в V-области создаются до фактического вызова FC.

В случае типов параметра, таких как TIMER, COUNTER и BLOCK_хх, параметр блока содержит номер (16-разрядное число, располагающееся с выравниванием влево в 32-разрядном параметре) вместо указателя на область.

В случае, если тип параметра POINTER, то параметр обрабатывается точно также, как данные сложных типов.

В случае, если тип параметра ANY, то редактор создает 10-байтовый ANY-указатель в V-области, который указывает на любую (ANY) переменную. Такой же подход распространяется на сложные типы данных.

Редактор делает исключение в случае, если Вы используете для параметра блока типа ANY фактический параметр, который размещается в области временных локальных данных и относится к типу данных ANY. В этом случае редактор не создает никаких ANY-указателей, а вместо этого направляет "указатель на область" (area-pointer) (параметр блока) прямо на фактический параметр (в этом случае

ANY-указатель может быть модифицирован в процессе выполнения программы, см. пункт 6.3.5 "Переменная" ANY-указатель).

6.3.4. Хранение параметров в функциональных блоках

Редактор сохраняет параметры для функционального блока в его экземплярном блоке данных. При вызове функционального блока редактор создает код, который обеспечивает копирование значений фактических параметров в экземплярный блок данных перед "фактическим" вызовом функционального блока, а затем копирует значения этих параметров назад из экземплярного блока в фактические параметры после вызова блока. При просмотре скомпилированного блока Вы не можете видеть этот код, создаваемый редактором, но Вы можете заметить его присутствие по косвенным признакам - по загрузке памяти. В экземплярном блоке данных параметры блока хранятся либо как значения (в виде 16-разрядных чисел), либо как указатель, указывающий на фактический параметр (см. табл. 6.12). Если параметр хранится как значение, то требуемая для этого память зависит от типа данных параметра блока. Так, число занимает два байта, указатели занимают 6 байтов (DB-указатели) или 10 байтов (ANY-указатели).

Таблица 6.12. Доступ к параметрам функционального блока

Тип данных	INPUT	IN_OUT	OUTPUT
Простой тип	значение	значение	значение
Сложный тип	значение	DB-указатель	значение
TIMER, COUNTER и BLOCK	номер	недопустимый	недопустимый
POINTER	DB-указатель	DB-указатель	недопустимый
ANY	ANY-указатель	ANY-указатель	недопустимый

При копировании значений фактических параметров сложных типов в экземплярный блок данных (входные параметры) и при копировании значений параметров из экземплярного блока в фактические параметры (выходные параметры) редактор использует системную функцию SFC 20 (BLKMOV), параметры которой он встраивает в область временных локальных данных вызывающего блока.

Перед текущим вызовом функционального блока и после такого вызова производится копирование и передача параметров блока в экземплярный блок данных с использованием последовательности операторов кода, вносимого редактором при компиляции. Перед текущим вызовом функционального блока производится копирование входных и входных-выходных параметров блока, а после вызова функционального блока производится копирование входных/выходных и выходных параметров блока. Поэтому здесь действует правило, согласно которому Вы можете только проверить входные параметры и сделать запись в выходные параметры. Если Вы, к примеру, передаете (новое) значение во входной параметр, то текущее значение фактического параметра будет потеряно.

Если Вы загружаете выходной параметр, это значит, что Вы загружаете (старое) значение из экземплярного блока данных, а не значение фактического параметра.

Так как параметры блока хранятся в экземплярном блоке данных, то нет необходимости инициализировать их всякий раз при вызове функционального блока. Если не была проведена инициализация параметров блока, то в программе будут использоваться "старые" значения входных и входных-выходных параметров блока, а что касается выходных параметров блока, то Вы сможете считать их значения, относящиеся к другой точке выполняемой программы. Вне функционального блока Вы можете получить доступ к переменным, которые хранятся в экземплярном блоке данных, таким же способом, как и к переменным в блоке глобальных данных (с использованием символического имени блока данных и имени параметра блока). Это касается также и статических локальных данных.

Если Вы используете временную локальную переменную типа ANY в ANY-параметре, то редактор копирует содержание этой переменной в ANY-указатель (в параметр блока) в экземплярном блоке данных.

6.3.5. "Переменная" ANY-указатель

ANY-параметры могут быть параметризованы только такими областями данных или переменными, которые уже определены на этапе компиляции.

Пример: копирование переменной в область данных с помощью системной функции SFC 20:

```
CALL SFC 20
SRCBLK := "ReceiveMailbox".Data
DSTBLK := P#DB63.DBW0.0 BYTE 8
RET_VAL := SFC20Error
```

Во время работы программы можно модифицировать или переопределить переменную или область данных, так как редактор использует фиксированный ANY-указатель на фактический параметр во временных локальных данных (см. далее в данном разделе).

Но есть одно исключение, которое касается случая, когда фактический параметр располагается во временных локальных данных и имеет тип ANY. Тогда редактор не создает никакого ANY-указателя, вместо этого он интерпретирует эту ANY-переменную как ANY-указатель на фактический параметр. Это означает, что ANY-переменная должна иметь такую же структуру как ANY-указатель.

Во время работы программы Вы можете модифицировать такие ANY-переменные во временных локальных данных и тем самым специфицировать другие фактические параметры, соответствующие ANY-параметрам.

Для применения таких "переменных" ANY-указателей Вы должны выполнить описанное ниже.

- Применение временной локальной переменной типа ANY (имя ANY-переменной может быть выбрано произвольным в допустимых рамках для локальных переменных блока):

```
VAR_TEMP
  ANY_POINTER : ANY
END_VAR
```

- Инициализация ANY-переменной значениями, указанными в таблице 6.13.
- Инициализация ANY-параметра, например, в блоке SFC 20:

```
CALL SFC 20
  SRCBLK := "ReceiveMailbox".Data
  DSTBLK := P#DB63.DBW0.0 BYTE 8
  RET_VAL := SFC20Error
```

Данная процедура не ограничивается только функциональным блоком SFC 20; Вы можете применять ее в отношении всех ANY-параметров любых блоков.

Таблица 6.13.

Адрес, известный во время выполнения программы	Адрес, неизвестный во время выполнения программы
	LAR1 P#ANY_POINTER;
L W#16#1002	L W#16#1002
T LW 0	T LW[AR1,P#0.0]
L 16	L 16
T LW 2	T LW[AR1,P#2.0]
L 63	L 63
T LW 4	T LW[AR1,P#4.0]
L P#DBB0.0	L P#DBB0.0
T LD 6	T LD[AR1,P#6.0]

Пример: пусть необходимо создать блок, копирующий одну область данных в другую. Исходная область и область назначения должны быть параметризуемыми. Мы используем функцию SFC 20 для копирования.

Блок FC имеет следующие параметры:

```
QDB : INT    //Исходный блок данных
SSTA : INT   //Начальный адрес исходных данных
NUMB : INT   //Число байтов
DDB : INT    //Блок данных назначения
DSTA : INT   //Начальный адрес области назначения
```

Возвращаемое значение функции должно содержать сообщение об ошибках функции SFC 20. Кроме того бит слова состояния BR устанавливается в состояние "0" в случае ошибки. Для работы с локальными данными блока достаточно использовать две ANY-переменные: одну - как указатель для исходной области, а вторую - как указатель для области назначения:

```
SANY : ANY //ANY-указатель исходной области
DANY : ANY //ANY-указатель области назначения
```

Так как мы знаем адреса ANY-указателей во временных локальных данных, мы можем запрограммировать их, используя абсолютную адресацию, например, при подготовке ANY-указателя области назначения:

```
L W#16#1002 // тип BYTE
T LW 0
```

```

L  NUMB      // число байтов
T  LW 2
L  QDB       // исходный DB
T  LW 4
L  SSTA      // начало исходных данных
SLD 3
OD  DW#16#8400_0000
T  LD 6

```

Указатель области назначения с начальным адресом LB 10 программируется аналогичным образом. Остается только инициализировать блок SFC 20:

```

CALL SFC 20
SRCBLK := SANY
DSTBLK := DANY
RET_VAL := RET_VAL

```

Значение функции RET_VAL для SFC 20 инициализируется значением RET_VAL нашей функции.

Таким образом, ANY-указателю может быть назначено любое значение, например, Вы можете менять тип в слове 2 или указатель на область, так что Вы можете адресовать любые переменные и области данных, например, область меркеров.

Примечание: если ANY-указатель, размещенный во временных локальных данных, указывает на переменную, которая также размещена во временных локальных данных вызывающего блока, то V-область должна быть введена как область-операнд, так как с точки зрения вызванного блока данная переменная размещена во временных локальных данных предшествующего блока.

Основная литература

1. Мюллер Ю. Регулирование на основе SIMATIC //Практическое пособие по регулированию на основе SIMATIC и PCS7. Пер. с нем. 2-е изд., 2002.
2. Бергер Х. Автоматизация посредством Step7 с использованием STL и SCL на программируемых контроллерах SIMATIC S7 – 300/400, 2001.

Дополнительная литература

Техническая документация фирмы Siemens (файлы с сайта <http://www.automation-drives.ru>)

- 1) S7-PLCSIM_V5_r.pdf
- 2) Step7.V53_HW_Config_r.pdf.
- 3) Step7.V53_Programming_r.pdf.